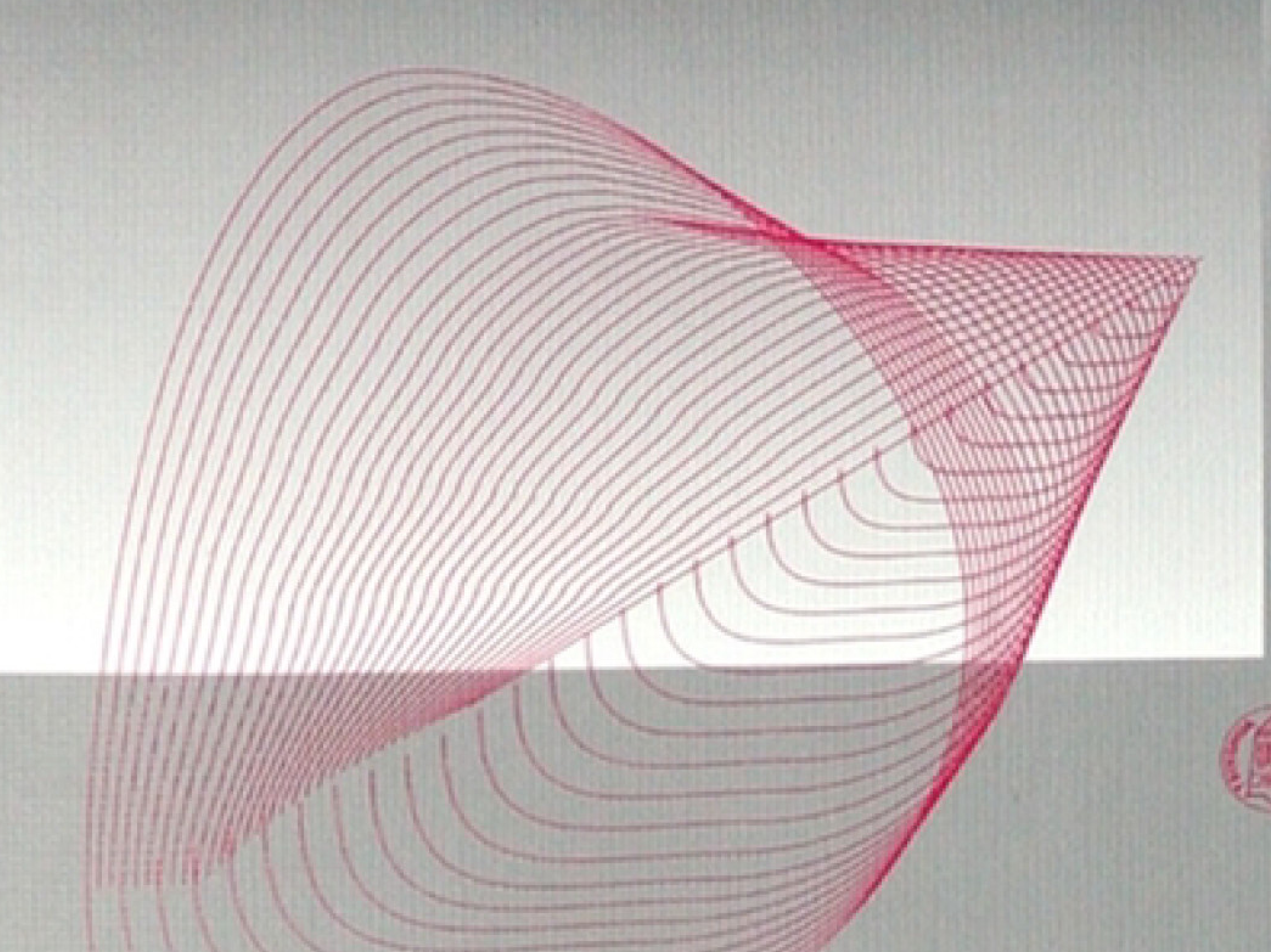
可下载教学资料 http://www.tup.tsinghua.edu.cn

21 世纪高等学校计算机教育实用规划教材

软件测试技术基础

陈汶滨 朱小梅
任冬梅 编著







清华大学出版社

21世纪高等学校计算机教育实用规划教材

软件测试技术基础

陈汶滨　朱小梅　任冬梅　编著

清华大学出版社
北京

内 容 简 介

本书针对高校计算机专业软件测试课程的需要而编写，主要介绍了软件测试的基础知识与应用技术。内容包含软件测试概述、软件测试方法与过程、黑盒测试、白盒测试、软件测试管理、自动化测试基础以及对主流测试工具软件 WinRunner、LoadRunner 和 JUnit 的详细介绍，并附有大量实际案例。本书能同时满足课堂理论教学与上机实践教学的需要，便于学生在学习过程中及时将理论知识运用于实际问题的解决，实用性较强。

全书讲解深入浅出，内容结构合理，适于高校计算机相关专业作为软件测试课程教材使用，同时也可作为软件测试人员的参考用书。

图书在版编目(CIP)数据

软件测试技术基础/陈汶滨，朱小梅，任冬梅编著. —北京：清华大学出版社，2008.7
(21世纪高等学校计算机教育实用规划教材)
ISBN 978-7-302-17493-6

Ⅰ. 软…　Ⅱ. ①陈… ②朱… ③任…　Ⅲ. 软件—测试—高等学校—教材　Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字(2008)第 058919 号

责任编辑：梁　颖　李玮琪
责任校对：时翠兰
责任印制：

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×260　**印　张**：13.25　**字　数**：320 千字
版　次：2008 年 7 月第 1 版　**印　次**：2008 年 7 月第1 次印刷
印　数：1～　000
定　价：　.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：

出版说明

随着我国高等教育规模的扩大以及产业结构调整的进一步完善，社会对高层次应用型人才的需求将更加迫切。各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，合理调整和配置教育资源，在改革和改造传统学科专业的基础上，加强工程型和应用型学科专业建设，积极设置主要面向地方支柱产业、高新技术产业、服务业的工程型和应用型学科专业，积极为地方经济建设输送各类应用型人才。各高校加大了使用信息科学等现代科学技术提升、改造传统学科专业的力度，从而实现传统学科专业向工程型和应用型学科专业的发展与转变。在发挥传统学科专业师资力量强、办学经验丰富、教学资源充裕等优势的同时，不断更新其教学内容、改革课程体系，使工程型和应用型学科专业教育与经济建设相适应。计算机课程教学在从传统学科向工程型和应用型学科转变中起着至关重要的作用，工程型和应用型学科专业中的计算机课程设置、内容体系和教学手段及方法等也具有不同于传统学科的鲜明特点。

为了配合高校工程型和应用型学科专业的建设和发展，急需出版一批内容新、体系新、方法新、手段新的高水平计算机课程教材。目前，工程型和应用型学科专业计算机课程教材的建设工作仍滞后于教学改革的实践，如现有的计算机教材中有不少内容陈旧（依然用传统专业计算机教材代替工程型和应用型学科专业教材），重理论、轻实践，不能满足新的教学计划、课程设置的需要；一些课程的教材可供选择的品种太少；一些基础课的教材虽然品种较多，但低水平重复严重；有些教材内容庞杂，书越编越厚；专业课教材、教学辅助教材及教学参考书短缺，等等，都不利于学生能力的提高和素质的培养。为此，在教育部相关教学指导委员会专家的指导和建议下，清华大学出版社组织出版本系列教材，以满足工程型和应用型学科专业计算机课程教学的需要。本系列教材在规划过程中体现了如下一些基本原则和特点。

（1）面向工程型与应用型学科专业，强调计算机在各专业中的应用。教材内容坚持基本理论适度，反映基本理论和原理的综合应用，强调实践和应用环节。

（2）反映教学需要，促进教学发展。教材规划以新的工程型和应用型专业目录为依据。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生知识、能力、素质协调发展创造条件。

（3）实施精品战略，突出重点，保证质量。规划教材建设仍然把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现工程型和应用型专业教学内容和课程体系改革成果的教材。

（4）主张一纲多本，合理配套。基础课和专业基础课教材要配套，同一门课程可以有多本具有不同内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

（5）依靠专家，择优选用。在制订教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主编。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平的以老带新的教材编写队伍才能保证教材的编写质量和建设力度，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21世纪高等学校计算机教育实用规划教材编委会

联系人：丁岭 dingl@tup.tsinghua.edu.cn

前　言

软件测试是一门非常崭新的学科，目前研究的内容还很不深入，仍然处于起步阶段。软件测试需要什么样的专业基础还没有定论，而且目前还没有一种很好的标准来衡量测试人员。但软件测试越来越受到软件公司的重视，软件测试工程师的作用也逐渐被人们所认可。这一点已经在像微软这样的国外大型软件企业中证实，在微软公司中，一个开发人员对应着一至两个测试人员。在国内，随着人们对软件本质的进一步认识，软件测试对于软件质量保证的作用已被软件企业所重视，软件测试在软件开发中的作用也越来越重要，软件测试的地位得到空前的提高。软件测试已经成为软件产业的新兴门类迅速发展起来，软件测试专业机构和组织在迅速发展，软件测试人才需求旺盛，测试职业的价值日益提升。

目前软件测试类教材研究还处于起步阶段，已有的其他同类教材大都侧重于理论介绍，对具体实践操作讨论很少。针对这种情况，本书在内容安排上既详细介绍了基础理论，又重点突出了实践应用，具体分为软件测试概述、软件测试方法与过程、黑盒测试、白盒测试、软件测试管理及自动化测试基础、WinRunner 测试工具、LoadRunner 测试工具、JUnit 共 8 章，系统全面地阐述了软件测试所涉及的基本概念、基本过程、基本方法和应用技术。其中第 1～5 章对软件测试基本理论与技术作了系统全面的讲解，深度把握适中，有助于帮助学生树立正确的软件测试概念，掌握测试基本原理；第 6～8 章则介绍了软件测试各阶段常用的主流测试工具软件的使用，给出了一个比较完整的软件测试流程解决方案，很好地将理论与实践结合在一起，能帮助学生在正确认识和理解软件测试理论知识的基础上掌握当前的主流测试技术，及时将理论知识运用于实际问题的解决，培养学生实际操作能力。

书中每章均附有习题，可通过习题练习巩固与加深所学知识。

本书适合高校计算机相关专业作为软件测试课程教材使用，同时也可作为软件测试人员的参考用书。

本书第 1～3 章由陈汶滨编写，第 4、5 章由朱小梅编写，第 6、7 章由任冬梅编写，第 8 章由周英、王申申和吕曼曼等 3 位研究生共同编写，他们也参与了大量的书稿整理和成文工作，黎明教授和周荣辉教授在百忙之中抽出时间对本书进行了审阅。本书在编写和修订过程中得到了西南石油大学计算机科学学院领导以及软件工程教研室各位老师的帮助，在此表示感谢。此外，本书在编写和修订时参考了一些软件测试文献和资料，使我们受益匪浅，特向其作者表示感谢。

鉴于本书作者水平有限，加之时间仓促，书中难免有疏漏和不足之处，敬请广大读者批评指正。

编　者

2008 年 1 月

目　录

第 1 章　软件测试概述

1.1　软件测试背景

随着计算机技术的迅速发展和广泛深入地应用，软件系统的规模和复杂性也与日俱增，软件中存在的缺陷与故障造成的各类损失也大大增加了，有的甚至会带来灾难性的后果。软件质量问题已成为所有使用软件和开发软件人员关注的焦点。而由于软件本身的特性，软件中的错误是不可避免的。不断改进的开发技术和工具只能减少错误的发生，但是却不可能完全避免错误。因此为了保证软件质量，必须对软件进行测试。软件测试是软件开发中必不可少的环节，是最有效的排除和防治软件缺陷的手段。

随着人们对软件测试重要性的认识越来越深刻，软件测试阶段在整个软件开发周期中所占的比重日益增大。大量测试文献表明，通常花费在软件测试和排错上的代价大约占软件开发总代价的 50%以上。现在有些软件开发机构将研制力量的 40%以上投入到软件测试之中；对于某些性命攸关的软件，其测试费用甚至高达所有其他软件工程阶段费用总和的 3～5 倍。美国微软公司软件测试人员是开发人员的 1.5～2.5 倍。

当软件业不断成熟，走入工业化阶段的同时，软件测试在软件开发领域的地位也越来越重要。

1.1.1　软件可靠性

已投入运用的软件质量的一个重要标志是软件可靠性。从实验系统所获得的统计数据表明，运行软件的驻留故障密度各不相同，与生命攸关的关键软件为每千行代码 0.01～1 个故障，与财务(财产)有关的关键软件为每千行代码 1～10 个故障，其他对可靠性要求相对较低的软件系统故障就更多了。然而，正是由于软件可靠性的大幅度提高才使得计算机得以广泛应用于社会的各个方面。

一个可靠的软件应该是正确的、完整的、一致的和健壮的。美国电气和电子工程师协会(IEEE)将软件可靠性定义为：系统在特定的环境下，在给定的时间内无故障地运行的概率。软件可靠性牵涉到软件的性能、功能性、可用性、可服务性、可安装性、可维护性以及文档等多方面特性，是对软件在设计、生产以及在它所预定环境中具有所需功能的置信度的一个度量，是衡量软件质量的主要参数之一。软件测试则是保证软件质量，提高软件可靠性的最重要手段。

1.1.2 软件缺陷

1. 软件缺陷案例

当今人类的生存和发展已经离不开各种各样的信息服务，为了获取这些信息，需要计算机网络或通信网络的支持，不仅需要计算机硬件等基础设施或设备，还需要各式各样的、功能各异的计算机软件。软件在电子信息领域里无处不在，然而，软件是由人编写开发的，是一种逻辑思维的产品，尽管现在软件开发者采取了一系列有效措施，不断地提高软件开发的质量，但仍然无法完全避免软件(产品)会存在各种各样的缺陷。下面介绍两个软件缺陷的案例，借此说明软件缺陷问题有时会造成相当严重的损失和灾难。

(1) 跨世纪“千年虫”问题

跨世纪“千年虫”问题，是一个非常著名的计算机软件缺陷问题。在20世纪末的最后几年中，全世界的各类硬件系统、软件系统和应用系统都为“千年虫”问题付出了巨大的代价。

20世纪70年代，程序员为了节约非常宝贵的内存资源和硬盘空间，在存储日期时只保留了年份的后两位数，如“1980”被存为“80”。他们采用这一措施的出发点主要是认为只有到了2000年程序在计算00或01这样的年份时才会出现问题，但在到达2000年时，程序早已不用或者修改升级了。然而，令这些程序员万万没有想到的是，他们的程序会被一直用到2000年，当2000年到来时，问题就出现了。计算机系统在处理2000年年份问题(以及与此年份相关的其他问题)时，软、硬件系统中存在的问题隐患被业界称为“千年虫”问题。

据不完全统计，从1998年年初全球就开始进行“千年虫”问题的大检查，特别是金融、保险、军事、科学、商务等领域花费了大量的人力、物力对现有的各种各样的程序进行检查、修改和更正，仅此项费用就达数百亿美元。

(2) Windows 2000 中文输入法漏洞

在安装微软的 Windows 2000 简体中文版的过程中，在默认情况下会同时安装各种简体中文输入法。随后这些装入的输入法可以在 Windows 2000 系统用户登录界面中使用，以便用户能够使用基于字符的用户表示和密码登录系统。然而，在默认安装的情况下，Windows 2000 中的简体中文输入法不能正确检测当前的状态，导致了在系统登录界面中提供了不应有的功能，即出现了下面的问题。

在 Windows 2000 用户登录界面中，当用户输入用户名时，用 Ctrl+Shift 组合键将输入法切换到全拼输入法状态下，同时在登录界面的屏幕左下角将会出现输入法状态条。用鼠标右击状态条并在出现的菜单中选择【帮助】项，将鼠标移到【帮助】项上，在弹出的选择项里选择【输入法入门】项，随后即弹出【输入法操作指南】帮助窗口。再用鼠标右击【选项】项，并选择【跳至 URL】项，此时将出现 Windows 2000 的系统安装路径并要求添入路径的空白栏。如果该操作系统安装在C盘上，在空白栏中填入“C:\WINDOWSNT\system32”，并单击【确定】按钮，在【输入法操作指南】右边的文本框里就会出现 C:\WINDOWSNT\system32 目录下的内容了，也就是说这样的操作成功地绕过了身份的验证，顺利地进入了系统的 system32 目录，当然也就可以进行各种各样的操作了。

此缺陷被披露后，微软公司推出了该输入法的漏洞补丁，并在 Windows 2000 Server

Pack2 以后的补丁中都包含了对该漏洞的修补,但对于没有安装补丁的用户来说,系统仍然处于不安全的状态之中。

2. 软件缺陷的定义和种类

上面实例中的软件问题在软件工程或软件测试中都被称为软件缺陷或软件故障。在不引起误解的情况下,不管软件存在问题的规模和危害的大小,由于都会产生软件使用上的各种障碍,所以将这些问题统称为软件缺陷。

软件缺陷,即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要。在 IEEE 1983 of IEEE Standard 729 中对软件缺陷下了一个标准的定义,如下:

从产品内部看,软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题;从外部看,软件缺陷是系统所需要实现的某种功能的失效或违背。因此软件缺陷就是软件产品中所存在的问题,最终表现为用户所需要的功能没有完全实现,没有满足用户的需求。

软件缺陷表现的形式有多种,不仅仅体现在功能的失效方面,还体现在其他方面。软件缺陷的主要类型通常有以下几种。

(1) 软件未达到产品说明书中已经标明的功能;

(2) 软件出现了产品说明书中指明不会出现的错误;

(3) 软件未达到产品说明书中虽未指出但应当达到的目标;

(4) 软件功能超出了产品说明书中指出的范围;

(5) 软件测试人员认为软件难以理解、不易使用,或者最终用户认为该软件使用效果不良。

为了对以上 5 条描述进行理解,这里以日常我们所使用的计算器内的嵌入式软件来说明上述每条定义的规则。

计算器说明书一般声称该计算器将准确无误地进行加、减、乘、除运算。如果测试人员或用户选定了两个数值后,随意按下了"+"号键,结果没有任何反应或得到一个错误的结果,根据第一条规则,这是一个软件缺陷;如果得到错误答案,根据第一条规则,同样是软件缺陷。

假如计算器产品说明书指明计算器不会出现崩溃、死锁或者停止反应,而在用户随意按、敲键盘后,计算器停止接受输入或没有了反应,根据第二条规则,这也是一个软件缺陷。

若在测试过程中发现,因为电池没电而导致了计算不正确,但产品说明书未能指出在此情况下应如何进行处理,根据第三条规则,这也应算作软件缺陷。

若在进行测试时,发现除了规定的加、减、乘、除功能之外,还能够进行求平方根的运算,而这一功能并没有在说明书的功能中规定,根据第四条规则,这也是软件缺陷。

第五条的规则说明了无论测试人员或者是最终用户,若发现计算器某些地方不好用,比如,按键太小,显示屏在亮光下无法看清等,也都应算作是软件缺陷。

软件缺陷一旦被发现,就要设法找出引起这个缺陷的原因,分析对产品质量的影响,然后确定软件缺陷的严重性和处理这个缺陷的优先级。各种软件缺陷所造成的后果是不同的,有的仅仅是不方便,有的则可能是灾难性的。一般来说,问题越严重的,其优先级越高,越要得到及时的纠正。软件公司对缺陷严重性级别的定义不尽相同,但一般可概括为以下

几种。

(1) 致命的：致命的错误，造成系统或应用程序崩溃、死机、系统悬挂，或造成数据丢失、主要功能完全丧失等。

(2) 严重的：严重错误，指功能或特性没有实现、主要功能丧失、会导致严重的问题或致命的错误声明。

(3) 一般的：不太严重的错误，这样的软件缺陷虽然不影响系统的基本使用，但没有很好地实现功能，没有达到预期效果。如次要功能丧失、提示信息不太准确、用户界面差、操作时间长等。

(4) 微小的：一些小问题，对功能几乎没有影响，产品及属性仍可使用，如有个别错误字、文字排列不整齐等。

除了这 4 种以外，有时需要“建议”级别来处理测试人员所提出的建议或质疑，对建议程序做适当的修改，来改善程序运行状态，或对设计不合理、不明白的地方提出质疑。

3. 软件缺陷的产生

软件缺陷的产生是不可避免的，那么造成软件缺陷的原因是什么呢？通过大量的测试理论研究及测试实践经验的积累，软件缺陷产生的主要原因可以被归纳为以下几种类型。

(1) 需求解释有错误；

(2) 用户需求定义错误；

(3) 需求记录错误；

(4) 设计说明有误；

(5) 编码说明有误；

(6) 程序代码有误；

(7) 其他，如：数据输入有误，问题修改不正确。

由此可见，造成软件缺陷的原因是多方面的。经过软件测试专家们的研究发现，大多数的软件缺陷并非来自编码过程中的错误，从小项目到大项目都基本上证明了这一点。因为软件缺陷很可能是在系统详细设计阶段、概要设计阶段，甚至是在需求分析阶段就存在着问题，即使是针对源程序进行的测试所发现的故障的根源也可能存在于软件开发前期的各个阶段。大量的事实表明，导致软件缺陷的最大原因是软件需求说明书，也是软件缺陷出现最多的地方。

在多数情况下，软件需求说明书写得不明确、不清楚、描述不全面，或者在软件开发过程中对需求、产品功能经常更改，或者开发小组的人员之间没有很好地进行交流与沟通，没有很好地组织开发与测试流程。因此，制作软件产品开发计划是非常重要的，如果计划没有做好，软件缺陷就会出现。

软件缺陷产生的第二大来源是设计方案，这是实施软件计划的关键环节。

编程排在第三位。许多人认为软件测试主要是找程序代码中的错误，这是一个认识的误区。经统计，因编写程序代码引入的软件缺陷大约仅占缺陷总数的 7%。

4. 软件缺陷的修复费用

软件通常要靠有计划、有条理的开发过程来建立。从前面的讨论可知，缺陷并不只是在编程阶段产生，在需求分析和设计阶段同样会产生。也许一开始只是一个很小范围内的潜在错误，但随着产品开发工作的进行，小错误会扩散成大错误，修改后期发现的错误所做的

工作要大得多。如果错误不能及早发现，那只可能造成越来越严重的后果。缺陷发现或解决得越迟，成本就越高。Boehm 在 *Software Engineering Economics*（1983 年）一书中曾经写到，平均而言，如果在需求阶段修正一个错误的代价是 1，那么在设计阶段就是它的 3～6 倍，在编程阶段是它的 10 倍，而到了产品发布出去时，这个数字就是 40～1000 倍。修正错误的代价不是随时间线性增长，而几乎是成指数级增长。

所以，测试人员应当把“尽早和不断地测试”作为其座右铭，从需求分析时就介入进去，尽早发现和改正错误。

1.1.3 软件测试发展与现状

20 世纪五六十年代，软件测试相对于开发工作仍然处于次要位置，测试理论和方法的发展都比较缓慢。除了极关键软件系统外一般测试都不完备，导致大量包含大大小小缺陷的软件投入运行，一旦暴露即带来不同程度的严重后果，例如早年火星探测运载火箭因控制程序中错写了一个逗号而爆炸。

随着人们对软件测试重要性的认识和软件技术的不断成熟和完善，20 世纪 70 年代以后软件测试的规模和复杂度日益加大，并逐渐形成了一套完整的体系，开始走向规范化。1982 年在美国北卡罗来纳州大学召开了首次软件测试技术会议，这是软件测试与软件质量研究人员和开发人员的第一次聚会，成为软件测试技术发展的一个重要里程碑。此后，测试理论、测试方法进一步完善，从而使软件测试这一实践性很强的学科成为有理论指导的学科。

不过尽管软件测试技术与实践都有了很大进展，但是就目前软件工程发展状况而言，软件测试仍然是较为薄弱的一个方面。而国内软件测试工作相对于国外起步较晚，与一些发达国家相比还存在一定差距，因此对于国内软件企业来说，需要进一步提高对软件测试重要性的认识，研究与采用先进的测试管理与应用技术，建立完善的软件质量保证的管理体系。

1.2 软件测试基础理论

1.2.1 软件测试定义

1. 软件测试的定义

软件测试就是在软件投入运行前，对软件需求分析、设计规格说明和编码实现的最终审查，它是软件质量保证的关键步骤。

根据著名软件测试专家 G. J. Myers 的观点，“软件测试是为了发现错误而执行程序的过程”。根据该定义，软件测试是根据软件开发各个阶段的规格说明和程序的内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果），并利用这些测试用例运行程序以及发现错误的过程，即执行测试步骤。测试是采用测试用例执行软件的活动，它有两个显著目标：找出失效或演示正确的执行。

其中，测试用例是为特定的目的而设计的一组输入、执行条件和预期结果，它是执行测

试的最小实体。

测试步骤详细规定了如何设置、执行、评估特定的测试用例。

除此之外,G. J. Myers 还给出了以下与测试相关的 3 个重要观点。

(1) 测试是为了证明程序有错,而不是证明程序无错误;

(2) 一个好的测试用例是在于它能发现至今未发现的错误;

(3) 一个成功的测试是发现了至今未发现的错误的测试。

在这一测试定义中,明确指出"寻找错误"是测试的目的,相对于"程序测试是证明程序中不存在错误的过程",Myers 的定义是对的。因为把证明程序无错当作测试的目的不仅是不正确的、完全做不到的,而且对于做好测试工作没有任何益处,甚至是十分有害的。因此从这方面讲,可以接受 Myers 的定义以及它所蕴含的方法观和观点。不过,这个定义也有局限性,它将测试定义规定的范围限制得过于狭窄,测试工作似乎只有在编码完成以后才能开始。更多专家认为软件测试的范围应当更为广泛,除了要考虑测试结果的正确性以外,还应关心程序的效率、可适用性、维护性、可扩充性、安全性、可靠性、系统性能、系统容量、可伸缩性、服务可管理性、兼容性等因素。随着人们对软件测试更广泛、深刻的认识,可以说对软件质量的判断决不只限于程序本身,而是整个软件研制过程。

综上所述,对于软件测试可以作出如下定义:软件测试是为了尽快尽早地发现在软件产品中所存在的各种软件缺陷而展开的贯穿整个软件开发生命周期,对软件产品(包括阶段性产品)进行验证和确认的活动过程。

2. 软件测试的基本问题

一个软件生命周期包括制定计划、需求分析定义、软件设计、程序编码、软件测试、软件运行、软件维护、软件停用等 8 个阶段。

软件测试的根本目的是为了保证软件质量。ANSI/IEEE Std 729-1983 文件中,软件质量概念被定义为"与软件产品满足规定的和隐含的需求的能力有关的特征或特征的全体"。软件质量反映在以下 3 个方面。

(1) 软件需求是度量质量的基础。

(2) 在各种标准中定义开发准则,用来指导软件人员用工程化的方法来开发软件。

(3) 往往会有一些隐含的需求没有明确地提出。如果软件只满足那些精确定义的需求,而没有满足那些隐含的需求,软件质量也不能得到保证。

软件质量内涵包括:正确性、可靠性、可维护性、可读性(文档、注释)、结构化、可测试性、可移植性、可扩展性、用户界面友好性、易学、易用、健壮性。

软件测试的对象:软件测试不仅仅是对程序的测试,而是贯穿于软件定义和开发的整个过程。因此,软件开发过程中产生的需求分析、概要设计、详细设计以及编码等各个阶段所得到的文档,包括需求规格说明书、概要设计规格说明书、详细设计规格说明书以及源代码,都是软件测试的对象。

软件测试设计的关键问题包括以下四个方面。

(1) 测试由谁来执行。

通常软件产品的开发设计包括开发者和测试者两种角色。开发者通过开发而形成产

品,主要工作是以上所列的分析、设计、编码、调试或文档编制等。测试者通过测试来检查产品中是否存在缺陷,包括根据特定目的而设计测试用例、构造测试、执行测试和评价测试结果等。通常的做法是开发者(机构或组织)负责完成自己代码的单元测试,而系统测试则由一些独立的测试人员或专门的测试机构进行。

(2) 测试什么。

测试经验表明,通常表现在程序中的故障,并不一定是由编码所引起。它可能是在详细设计阶段、概要设计阶段,甚至是需求分析阶段的错误所致。即使对源程序进行测试,所发现故障的根源也可能是在开发前期的某个阶段。要排除故障、修正错误也必须追溯到前期的工作。事实上,软件需求分析、设计和实施阶段是软件故障的主要来源。

(3) 什么时候进行测试。

测试可以是一个与开发并行的过程,还可以是在开发完成某个阶段任务之后的活动或者是开发结束之后的活动,即模块开发结束之后可以进行测试,也可以推迟在各模块装配成为一个完整的程序之后再进行测试。开发经验表明,随着开发不断深入,没有进行测试的模块对整个软件的潜在破坏作用更明显,因此,测试应尽早和不断地进行。

(4) 怎样进行测试。

软件"规范"说明了软件本身应该达到的目标,程序"实现"则是对应各种输入如何产生输出结果的算法。换言之,规范界定了一个软件要做什么,而程序实现则规定了软件应该怎样做。对软件进行测试就是根据软件的功能规范说明和程序实现,利用各种测试方法,生成有效的测试用例,对软件进行测试。

要实现软件质量保证,主要有两种途径:首先通过贯彻软件工程各种有效的技术方法和措施使得尽量在软件开发期间减少错误,其次就是通过分析和测试软件来发现和纠正错误。因此,软件测试是软件质量的重要保证。

对于一个系统做的测试越多,就越能确保它的正确性。然而,软件的测试通常不能保证系统的运行百分之百的正确。因此,软件测试在确保软件质量方面的主要贡献在于它能发现那些一开始就应避免的错误。软件质量保证的使命首先是避免错误。

1.2.2 软件测试基本理论

1. 软件测试的目的

从历史的观点来看,测试关注执行软件来获得软件在可用性方面的信心并且证明软件能够满意地工作,这引导测试把重点投入在检测和排除缺陷上。现代的软件测试沿用了这个观点,同时,还认识到许多重要的缺陷主要来自于对需求和设计的误解、遗漏和不正确。因此,早期的同行评审被用于帮助预防编码前的缺陷。证明、检测和预防已经成为一个良好测试的重要目标。

(1) 证明:获取系统在可接受风险范围内可用的信心;尝试在非正常情况和条件下的功能和特性;保证一个工作产品是完整的并且可用或可被集成。

(2) 检测:发现缺陷、错误和系统不足;定义系统的能力和局限性;提供组件、工作产品和系统的质量信息。

（3）预防：澄清系统的规格和性能；提供预防或减少可能制造错误的信息；在过程中尽早检测错误；确认问题和风险，并且提前确认解决这些问题和风险的途径。

2. 软件测试的原则

软件测试的基本原则是站在用户的角度，对产品进行全面测试，尽早、尽可能多地发现缺陷，并负责跟踪和分析产品中的问题，对不足之处提出质疑和改进意见。零缺陷是一种理想，足够好是测试的原则。

如果进一步去研究测试的原则，会发现在软件测试过程中，应注意和遵循的原则可以概括为以下 10 项。

（1）测试不是为了证明程序的正确性，而是为了证明程序不能工作。正如 Myers 所说，测试的目的是证伪而不是证真。事实上，证明程序的正确性是不可能的。一个大型的集成化的软件系统不能被穷尽测试以遍历其每条路径，而且即使遍历了所有的路径，错误仍有可能隐藏。做测试是为了尽可能地发现错误。

（2）测试应当有重点。因为时间和资源是有限的，不可能无休止地进行测试。测试的重点选择需要根据多个方面考虑，包括测试对象的关键程度，可能的风险，质量要求等。这些考虑与经验有关，随着实践经验的增长，判断也会更有效。

（3）事先定义好产品的质量标准。只有建立了质量标准，才能根据测试的结果，对产品的质量进行分析和评估。同样，测试用例应确定预期输出结果。如果无法确定测试结果，则无法进行校验。必须用事先精确对应的输入数据和输出结果来对照检查当前的输出结果是否正确，做到“有的放矢”。

（4）软件项目一启动，软件测试也就开始，而不是等到程序写完才开始进行测试。测试是一个持续进行的过程，而不是一个阶段。在代码完成之前，测试人员要参与需求分析、系统或程序设计的审查工作，而且要准备测试计划、测试用例、测试脚本和测试环境。测试计划可以在需求模型一完成就开始，详细的测试用例定义可以在设计模型被确定后开始。

（5）穷举测试是不可能的。即使一个大小适度的程序，其路径排列的数量也非常大，因此在测试中不可能运行路径的每一种组合。然而，充分覆盖程序逻辑，并确保程序设计中使用的所有条件都达到是有可能的。

（6）第三方进行测试会更客观、更有效。程序员应避免测试自己的程序，为达到最佳的效果，应由第三方来进行测试。测试是带有“挑剔性”的行为，心理状态是测试自己程序的障碍。同时对于需求规格说明的理解产生的错误也很难在程序员本人测试时被发现。

（7）软件测试计划是做好软件测试工作的前提。在进行实际测试之前，应制定良好的、切实可行的测试计划并严格执行，特别要确定测试策略和测试目标。

（8）测试用例是设计出来的，不是写出来的，所以要根据测试的目的，采用相应的方法去设计测试用例，从而提高测试的效率，更多的发现错误，提高程序的可靠性。除了检查程序是否做了它应该做的事，还要看程序是否做了它不该做的事。不仅应选用合理的输入数据，对于非法的输入也要设计测试用例进行测试。

（9）对发现错误较多的程序段，应进行更深入的测试。一般来说，一段程序中已发现的错误数越多，其中存在的错误概率也就越大。

（10）重视文档，妥善保存一切测试过程文档。测试计划、测试用例、测试报告都是检查

整个开发过程的主要依据，有利于今后流程改进，同时也是测试人员的智慧结晶和经验积累，对新人或今后的工作都有指导意义。

3. 测试在开发各个阶段的任务

(1) 项目规划阶段：负责从单元测试到系统测试的整个测试阶段的监控。

(2) 需求分析阶段：确定测试需求分析、系统测试计划的制定，评审后成为管理项目。测试需求分析是对产品生命周期中测试所需的资源、配置、每阶段评判通过的规约；系统测试计划则是依据软件的需求规格说明书，制定测试计划和设计相应的测试用例。

(3) 详细设计和概要设计阶段：确保集成测试计划和单元测试计划完成。

(4) 编码阶段：由开发人员进行自己负责部分的代码测试。在项目较大时，由专人进行编码阶段的测试任务。

(5) 测试阶段(单元测试、集成测试、系统测试)：依据测试代码进行测试，并提交相应的测试报告和测试结束报告。

4. 测试信息流

测试信息流如图 1.1 所示，测试过程中需要两类输入信息。

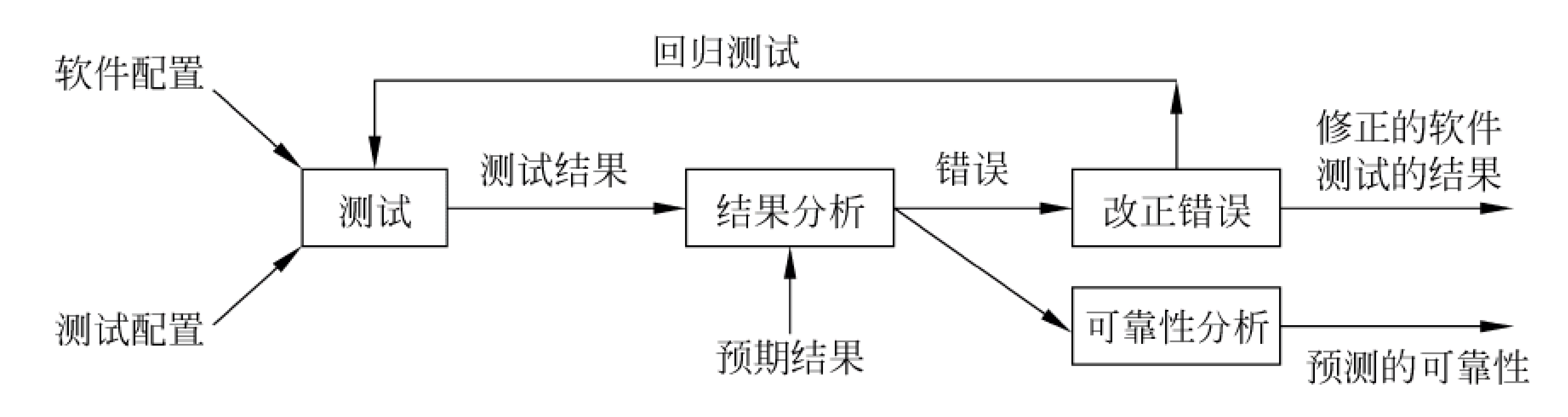


图 1.1　测试信息流图

(1) 软件配置：指测试对象。通常包括软件需求规格说明、软件设计规格说明、源代码等。

(2) 测试配置：通常包括测试计划、测试步骤、测试用例以及实施测试的测试程序、测试工具等。

对测试结果与预期的结果进行比较以后，即可判断是否存在错误，决定是否进入排错阶段，进行调试任务。由于修改可能会带来新的问题，故需要对修改以后的程序重新测试，即进行回归测试。

通常根据出错的情况得到出错率来预计被测试软件的可靠性，这将对软件运行后的维护工作有重要价值。

5. 软件测试停止标准

因为无法判定当前发现的故障是否为最后一个故障，所以决定什么时候停止测试是一件非常困难的事。受经济条件的限制，测试最终要停止。在实际工作中，常用的停止测试标准有以下 5 类。

(1) 第一类标准：测试超过了预定的时间，停止测试。

(2) 第二类标准：执行了所有测试用例但没有发现故障，停止测试。

(3) 第三类标准：使用特定的测试用例设计方法作为判断测试停止的基础。

(4) 第四类标准：正面指出测试停止的要求，比如发现并修改 70 个软件故障。

(5) 第五类标准：根据单位时间内查出故障的数量决定是否停止测试。

第一类标准意义不大,因为即便什么都不干也能满足这一条,这不能用来衡量测试的质量。

第二类标准同样也没有什么指导作用,因为它客观上鼓励人们编制查不出故障的测试用例。像上面所讨论的那样,人是有很强工作目的性的。如果告诉测试人员测试用例失败之时就是他完成任务之时,那他会不自觉地以此为目的去编写测试用例,回避那些更有用的、能暴露更多故障的测试用例。

第三类标准把使用特定的测试用例设计方法作为判断测试停止的基础。比如,可以定义测试用例的设计必须满足以下两个条件,作为模块测试停止的标准,即条件覆盖准则、边界值分析,并且由此产生的测试用例最终全部失败。尽管这类标准比前两个标准优越,但它只给出了一个测试用例设计的方法,并不是一个确定的目标。只有测试人员确实能够成功地运用测试用例设计的方法时,才能应用这类标准,并且这类标准只对某些测试阶段适用。

第四类标准正面指出了停止测试的要求,将其定义为查出某一预定数目的故障。它虽然加强了测试的定义,但存在两个方面的问题:如何知道将要查出的故障数;过高或过低估计故障总数。

第五类标准看上去很容易,但在实际使用中要用到很多判断和直觉。它要求人们用图表表示某个测试阶段中单位时间检查出的故障数量,通过分析图表,确定应继续进行测试还是结束这一测试阶段而开始下一测试阶段。

最好的停止测试标准或许是将上面讨论的几类标准结合起来。因为大部分软件开发项目在单元测试阶段并没有正式地跟踪查错过程,所以这一阶段最好的停止测试标准可能是第一类。对于集成测试和系统测试阶段,停止测试的标准可以是查出了预定数量的故障和达到了一定的测试期限,但还要分析故障时间图,只有当该图指明这一阶段的测试效率很低时才能停止测试。

1.2.3 软件测试技术概要

1. 软件测试的策略

任何实际的测试,都不能够保证被测软件中不存在遗漏的缺陷。为了最大程度地减少这种遗漏,同时也为了最大限度地发现已经存在的错误,在测试实施之前,软件测试工程师必须确定将要采用的软件测试策略和方法,并以此为依据制定详细的测试案例。一个好的软件测试策略和方法,必将给软件测试带来事半功倍的效果,它可以充分利用有限的人力和物力资源,高效率、高质量地完成测试。

软件测试的策略就是指测试将按照什么样的思路和方式进行。通常,针对代码的软件测试要经过单元测试、集成测试、确认测试、系统测试以及验收测试。

(1) 单元测试

单元测试也称为模块测试,是在软件测试当中进行的最低一级测试活动,它测试的对象是软件设计的最小单元。在面向过程的结构化程序中,如 C 程序,其测试的对象一般是函数或子过程。在面向对象的程序中,如 C++,单元测试的对象可以是类,也可以是类的成员函数。在第四代语言(4GL)中,单元测试的原则也基本适用,这时的单元被定义为一个菜单或显示界面。

单元测试的目的就是检测程序模块中的错误故障。

单元测试的任务是，针对每个程序模块，解决 5 个方面的问题：模块接口测试、模块局部数据结构测试、覆盖测试、出错处理检测、边界条件测试。

在对每个模块进行单元测试时，需要考虑各模块与周围模块之间的相互联系，因为每个模块在整个软件中并不是单一的。为模拟这一联系，在单元测试时，必须设计辅助测试模块，即驱动模块和桩模块，被测模块与这两个模块一起构成测试环境。

（2）集成测试

集成测试是按照设计要求将通过单元测试后的模块组合成一个整体测试的过程。因为程序在某些局部没有出现的问题，很可能在全局上暴露出来。

集成测试方法主要分为非增量式集成测试和增量式集成测试两种。

（3）确认测试

通过集成测试之后，独立的模块已经联系起来，构成一个完整的程序，其中各个模块之间存在的问题已被取消，即可以进入确认测试阶段。

所谓确认测试，是对照软件需求规格说明，对软件产品进行评估，以确认其是否满足软件需求的过程。

经过确认测试，应该为已开发的软件做出结论性的评价，这无非存在两种情况：其一，经过检验，软件功能、性能及其他方面的要求都已满足软件需求规格说明的规定，是一个合格的软件；其二，经过检验，发现与软件需求规格说明有相当的偏离，得到缺陷清单，这就需要开发部门和用户进行协商，找出解决的办法。

（4）系统测试

软件和硬件进行了一系列系统集成和测试，以保证系统各组成部件能够协调地工作。系统测试实际是针对系统中各个组成部分进行的综合测试。系统测试的目标不是要找出软件故障，而是要证明系统的性能。例如，确定安装过程是否会导致不正确的地方，确定系统或程序出现故障之后能否满足恢复性能要求，确定系统能否满足可靠性能要求等。

（5）验收测试

验收测试是将最终产品与最终用户的当前需求进行比较的全过程，是软件开发结束后软件产品向用户交付之前进行的最后一次质量检验活动。它解决软件产品是否符合预期的各项要求，用户是否接受等问题。验收测试是全面的质量检验并决定软件是否合格。

验收测试的主要任务是：明确验收测试通过的标准；确定验收计划、方式并对其进行评审；确定测试结果的分析方法；设计验收测试的测试用例；执行验收测试，分析验收结果，决定是否通过验收。

2. 软件测试方法和技术

软件测试的方法和技术多种多样，可以从以下不同的角度加以分类。

（1）根据执行测试的主体不同，可分为人工测试和自动化测试；

（2）根据软件测试针对系统的内部结构还是针对具体的实现功能，可分为白盒测试和黑盒测试；

（3）根据软件测试是否执行程序而论，可分为静态测试和动态测试；

(4) 按照测试的对象进行分类,分为面向开发的单元测试、GUI 和捕获/回放测试、基于 Web 应用的测试、C/C++ /Java 应用测试、负载和性能测试、数据库测试、软件测试和 QA 管理等各类工具测试;

(5) 其他测试方法,如回归测试、压力测试、恢复测试、安全测试和兼容性测试等。

1.3 软件开发

一个软件产品的建立可能需要数十个、数百个甚至上千个小组成员各司其职,并且在严格的进度计划中合作。指定这些人做什么、如何交流、如何做决定是软件开发过程的几大部分。软件开发过程是软件工程中的重要内容,也是进行软件测试的基础。

1.3.1 软件产品组成

分析构成软件产品的各个部分并了解常用的一些方法,对正确理解具体的软件测试任务和测试过程十分有益。

一般来说,开发软件产品需要产品说明书、产品审查、设计文档、进度计划、其他公司同类软件产品情况、客户调查、易用性数据、软件代码等一些大多数软件产品用户不曾想过的内容。

软件行业用于描述制造出来并交付他人使用的软件产品的术语是“可提供的”。为了得到“可提供的”软件产品,需要付出各种各样大量的工作。

1. 客户需求

编写软件的目的是满足客户的需求。为了更好地满足要求,产品开发小组必须弄清楚客户的需求。这里的需求包括调查收集的详细信息,以前软件的使用情况及存在的问题,竞争对手的软件产品信息等。除此之外,还有收集到的其他信息,并对这些信息进行研究和分析,以便确定将要开发的软件产品应该具有哪些功能。

要从客户那里得到反馈意见,目前除了直接由开发组进行调查外,还需通过独立调查机构进行调查问卷活动获得有关的问题反馈。

2. 产品说明书

对客户要求的研究结果其实只是原始资料,无法描述要做的产品,只是确定哪些要做、哪些不做以及客户所需要的产品功能。产品说明书的作用就是对上述信息进行综合描述,并包括用户没有提出但软件产品本身必须实现的要求,从而针对产品进行定义并确定其功能。

产品说明书的格式千差万别。对某些软件产品,如金融公司、航天系统、政府机构、军事部门的特制软件,要采取严格的程序对产品说明书进行检查,检查内容十分详细,并且在整个产品说明书中是完全确定的。在非特殊情况下,产品说明书是不能随意发生变化的,软件开发工作组的任务是完全确定的。

但有一些编制不严格的开发团队,某些应用软件产品的说明书写得比较简单粗糙,这种做法虽然比较灵活,但存在目标不明确的潜在问题。

3. 进度表

软件产品的一个关键部分是进度表。随着项目的不断扩大和复杂性的增加,开发产品

需要大量的人力、物力，必须有某种机制来跟踪进度。制定进度的目标是明确哪些工作完成了，哪些工作没有完成，何时能够完成。通常应用 Gantt 图表来描述开发进度，如图 1.2 所示。

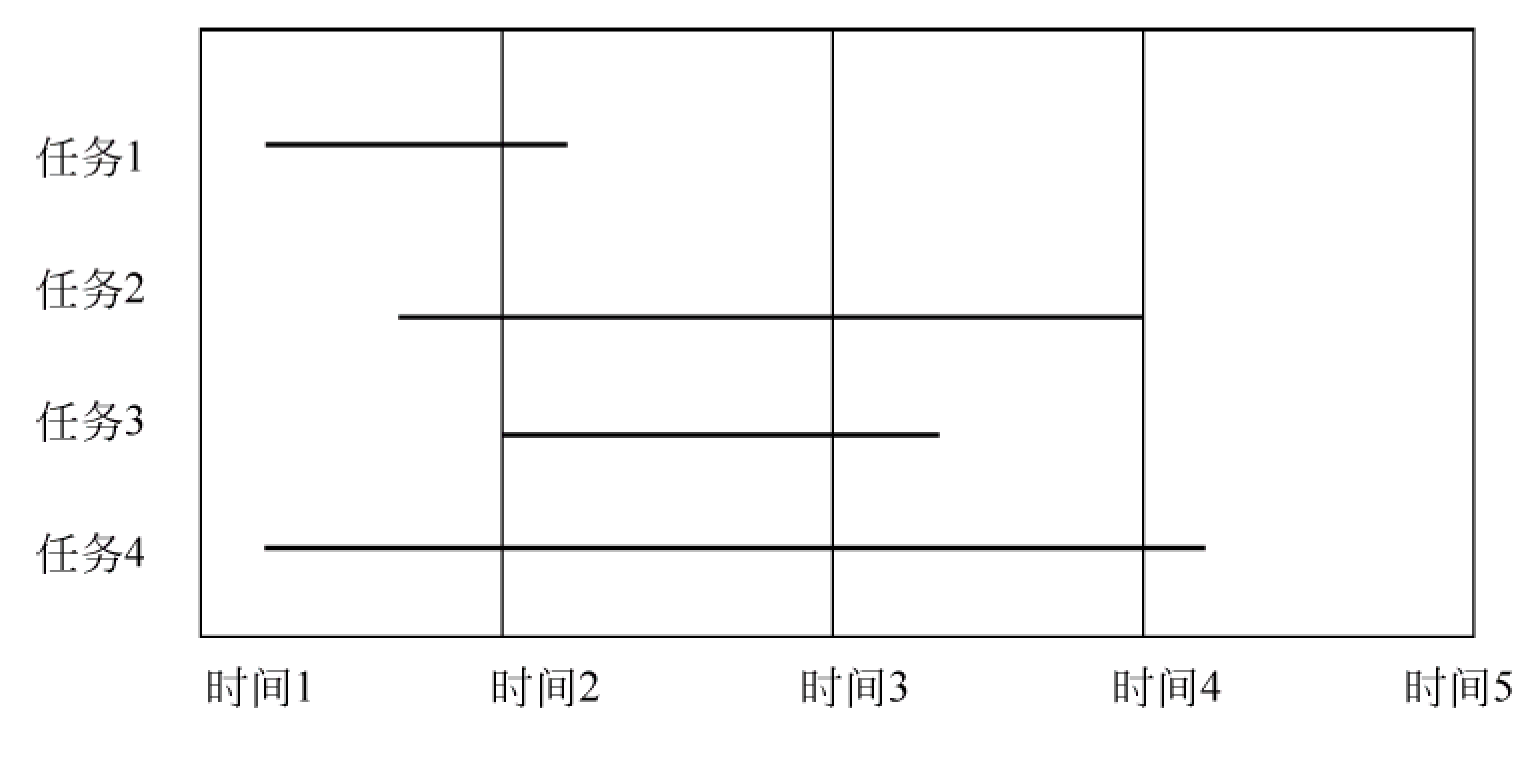


图 1.2　开发进度表

4. 设计文档

一个常见的错误观念是当程序员创建程序时，没有计划直接就开始编写代码。对于稍大一些的程序而言，就必须要有一个计划来编写软件的设计过程。

下面是一些常用软件设计文档的内容。

（1）构架。描述软件整体设计的文档，包括软件所有主要部分的描述以及相互之间的交互方式。

（2）数据流示意图。表示数据在程序中如何流动的正规示意图，有时被称为泡泡图。

（3）状态变化示意图。把软件分解为基本状态或者条件的另一种正规示意图，表示不同状态间变化的方式。

（4）流程图。用图形描述程序逻辑的传统方式。流程图现在虽不流行了，但是一旦投入使用，根据详细的流程图编写程序代码是很简单的。

（5）注释代码。在软件代码中嵌入有用的注释是极为重要的，这样便于维护代码的程序员轻松掌握代码的内容和执行方式。

5. 测试文档

测试文档是完整的软件产品的一部分。根据软件产品开发过程的需要，程序员和测试员必须对工作进行文档说明。

下面是一般测试文档所包含的内容。

（1）测试计划。描述用于验证软件是否符合产品说明书和客户需求的整体方案。

（2）测试案例。列举测试的项目，描述验证软件的详细步骤。

（3）软件缺陷报告。描述依据测试案例找出的问题。可以在纸上记录，但通常记录在数据库中。

（4）归纳、统计和总结。把生产过程转化为测试过程，采用图形、表格和报告等形式。

6. 软件产品的其他组成部分

软件产品不仅仅应当关心程序代码，还要关注各种各样的技术支持，这些部分通常由客户使用或查看，所以也需要进行测试。

下面列出软件产品除程序代码之外的其他各种组成。

(1) 帮助文件;

(2) 用户手册;

(3) 样本和示例;

(4) 标签;

(5) 产品支持信息;

(6) 图标和标志;

(7) 错误信息;

(8) 广告和宣传材料;

(9) 软件的安装;

(10) 软件说明文件;

(11) 测试错误提示信息。

1.3.2 开发人员角色

软件开发过程中,软件开发人员各司其职,根据职责的不同分为多种角色。

1. 项目经理

项目经理负责管理业务应用开发或者软件和系统开发项目。项目经理角色计划、管理和分配资源,确定优先级,协调用户和客户的交互。项目经理也要建立一系列的实践活动以确保项目工作产品的完整性和质量。

2. 业务分析人员

业务分析人员的任务是理解和描绘客户的需求,引导和协调用户和业务需求的收集和确认,使其文档化,组织系统的需求,或者向整个团队传达需求。

3. 架构师

架构师负责理解系统的业务需求,并创建合理、完善的系统体系架构。架构师也负责通过软件架构来决定主要的技术选择。典型的包括识别和文档化系统的重要架构方面,包括系统的需求、设计、实现和部署"视图"。

4. 数据设计人员

对于大多数的应用开发项目来说,用于持久存储数据的技术是关系型数据库。数据库架构师负责定义详细的数据库设计,包括表、索引、视图、约束、触发器、存储过程和其他的特定数据库用于存储、返回和删除持久性对象的结构。

5. 开发人员

开发人员通常负责设计和实现可执行的代码方案、测试开发出的组件和分析运行时情况,以去除可能存在的错误。有时开发人员还负责创建软件的体系架构或者使用快速应用开发工具。

6. 测试人员

系统测试人员负责制定测试计划并依照测试计划进行测试。这些测试包括功能性的测试(黑盒测试)和非功能性的测试(白盒测试)。测试人员需要良好的测试工具来辅助完成测试任务,自动化的测试工具将大幅度提高测试人员的工作效率和质量。

1.3.3 软件开发模式

1. 大棒模式

大棒模式的优点是简单,计划、进度安排和正规开发过程几乎没有。软件项目组成员的所有精力都花在开发软件和编写代码上,它的开发过程是非工程化的。

大棒模式的软件测试通常是在开发任务完成后进行,也就是说已形成了软件产品才进行测试。测试工作有的较为容易,有的则非常困难,这是因为软件及其说明书在最初就已经完成,待形成产品后,已经无法回头修复存在的问题,所以软件测试的工作只是向客户报告软件产品经过测试后发现的情况。

软件产品开发工作应当避免采用大棒模式作为软件开发的方法。

2. 边写边改模式

边写边改模式是项目小组在未刻意采用其他开发模式时常用的一种开发模式。它是在大棒模式基础上的一个进步,考虑到了软件产品的要求。

采用这种方式的软件开发通常最初只有粗略的想法,就进行简单的设计,然后开始较长的反复编写、测试和修复过程,在认为无法更精细地描述软件产品要求时就发布产品。

因为从开始就没有计划和文档的编制,项目组能够较为迅速地展现成果。因此,边写边改模式适合用在快速制作而且用完就扔的小项目上。

处于边写边改开发项目的软件测试人员将和程序员一起陷入可能是长期的循环往复的一个开发过程。通常,新的软件(程序)版本在不断地产生,而旧的版本的测试工作可能还未完成,新版本还可能又包含了新的或修改了的功能。

在进行软件测试工作期间,边写边改开发模式最有可能遇到。虽然它有缺点,但它是通向采用合理软件开发的路子,有助于理解更正规的软件开发方法。

3. 瀑布模式

瀑布模式是将软件生命周期的各项活动规定为按照固定顺序相连的若干个阶段性工作,形如瀑布流水,最终得到软件产品。

瀑布模式具有的优点是:易于理解;调研开发呈阶段性;强调早期计划及需求调查;确定何时能够交付产品及何时进行评审与测试。

但同时瀑布模式也存在一些缺点,如:需求调查分析只进行一次,不能适应需求的变化;顺序的开发流程,使得开发中的经验教训不能反馈到该项目的开发中去;不能反映出软件开发过程的反复性与迭代性;没有包含任何类型的风险评估;开发中出现的问题直到开发后期才能显露,因此失去了及早纠正的机会。

4. 快速原型模式

快速原型模式是一种以计算机为基础的系统开发方法,它首先构造一个功能简单的原型系统,然后通过对原型系统逐步求精,不断扩充完善得到最终的软件系统。原型就是模型,而原型系统就是应用系统的模型,它是待构筑的实际系统的缩小比例模型,但是保留了实际系统的大部分性能。这个模型可在运行中被检查、测试、修改,直到它的性能达到用户需求为止,因而这个工作模型很快就能转换成原样的目标系统。

快速原型模式的主要优点在于它是一种支持用户的方法,使得用户在系统生存周期的设计阶段起到积极的作用;它能减少系统开发的风险,特别是在大型项目的开发中,对项目

需求的分析难以一次完成，应用此方法效果明显。

5. 螺旋模式

螺旋模式是瀑布模式与边写边改模式演化、结合的形式，并加入了开发风险评估所建立的软件开发模式。

螺旋模式的主要思想是在开始时不必详细定义所有细节，而是从小开始，定义重要功能，尽量实现，接受客户反馈，进入下一阶段并重复上述过程，直到获得最终产品。

每一个螺旋（开发阶段）包括以下 6 个步骤。

（1）确定目标、选择方案和限制条件；

（2）指出方案风险并解决风险；

（3）对方案进行评估；

（4）进行本阶段的开发和测试；

（5）计划下一阶段；

（6）确定进入下一个阶段的方法。

螺旋开发模式中包含了一些瀑布模式（分析、设计、开发和开发步骤）、边写边改模式（每次盘旋上升）和大棒模式（从外界看）。该开发模式具有发现早、产品的来龙去脉清晰、成本相对低、测试从最初就参与各项工作的特点。该软件开发模式目前最常用，被广泛认为是软件开发的有效手段。

1.4 软件测试过程

美国 Carnegie Mellon 大学软件工程研究所（Software Engineering Institute）的 Don McAndrews 于 1997 年提出一个软件测试过程（Software Test Process）模型，该测试过程模型可用于确认测试、系统测试、验收测试或第三方软件测试过程。在此模型的基础上进行适当的扩充形成一个典型软件测试过程模型，该测试过程包括如下 6 个主要活动。

（1）测试计划。确定测试基本原则、生成测试概要设计。

（2）测试需求分析。

（3）测试设计。包括测试用例设计和测试规程规格说明。

（4）测试规程实现。

（5）测试执行。

（6）总结生成报告。

1. 测试计划

测试计划活动在软件开发项目的定义、规划、需求分析阶段执行，该项活动确定测试的基本原则并生成测试活动的高级计划。

测试计划在软件项目启动时开始，活动输入是项目进度表和系统/软件功能需求的描述（如软件的需求规格说明）。

测试计划包括以下步骤。

（1）项目经理和测试负责人共同参与测试过程相关的测试需求评审，主要包括：

- 进度表中各阶段的日程。
- 作为测试活动输入的相关合同中的可交付项。

- 项目进度表中针对测试活动而指定的时间。
- 客户指定的测试级别。
- 估计分配给测试活动的小时数。
- 客户在规格说明中指定的质量准则。

(2) 测试负责人制定一个针对该项目的测试策略，包括阶段、段、类型和级别。

(3) 测试负责人完成测试计划、用户规格说明、需求验证测试矩阵等测试策略文档，包括由客户规格说明定义的、从单元/集成测试到系统/验收测试的测试级别流程图。

(4) 测试负责人标示测试过程中产生的所有产品名称及交付日期。

(5) 项目经理和测试负责人标示项目功能需求的来源(如用户需求规格说明、功能规格说明、系统规格说明、合同或其他文档)，以便于实现需求追踪。

(6) 测试负责人审查用户需求规格说明中的功能需求以确定逻辑测试集。这一工作用于确定可重用策略，如果已存在类似的项目，测试负责人应当审查已有的测试产品以确定这些测试产品能否被重用。

(7) 测试负责人书写测试设计规格说明提纲。

(8) 测试负责人标示项目中将要进行的所有测试活动，包括测试准备、测试执行和测试后的活动并形成文档。

(9) 测试负责人在软件测试计划文档中描述测试活动。

(10) 测试负责人在项目进度表中标示测试活动、测试活动的起始和结束时间、风险和不可预见的费用。

(11) 测试负责人完成软件测试计划进度表，列出风险和不可预见费用并在测试活动描述中说明其度量。

(12) 基于当前可利用资源，测试项目负责人和项目经理安排测试人员和支持测试的人员并写入测试计划中。

(13) 测试负责人在软件测试计划文档中描述测试可交付项。

(14) 测试负责人和系统工程师定义测试环境(测试环境包括可用的硬件环境和必要的软件)并写入测试检查表中。

(15) 测试负责人、配置管理员和系统工程师定义测试控制规程(作为项目配置管理的组成部分)并写入测试计划。

(16) 测试负责人根据测试计划模板完成软件测试计划。测试计划包括风险和不可预见费用、暂停规则、恢复要求、缩略语列表等。

测试计划完成的标志是生成经过评审的软件测试计划文档，软件测试计划应获得客户的认可。测试规划完成后测试进入需求分析阶段。

2. 测试需求分析

在确定需求追踪矩阵、完成软件测试计划文档后即进入测试需求分析阶段。测试需求分析活动包括如下步骤。

(1) 测试负责人和测试工程师审查需求追踪矩阵中每个需求并为其确定测试方法。

(2) 测试负责人和测试工程师审查所有可测的需求并分配到测试设计规格说明中进行详细描述。

(3) 任何未在软件测试计划中标示的测试设计规格说明内容应当添加到需求追踪矩

阵中。

(4) 测试工程师生成关于测试需求指定到测试方法的报告供评审。

(5) 测试工程师生成关于可测试需求指定到测试设计规格说明的报告。

(6) 随着需求中问题的出现,测试负责人或测试工程师需书面描述问题并与项目经理讨论这些问题。如有必要,会生成基于缺陷的问题报告。

测试需求分析阶段的产品是被批准的需求测试矩阵。

3. 测试设计

完成需求测试矩阵和测试计划后,即可进入测试设计阶段,该阶段活动步骤如下。

(1) 测试工程师审查客户规格说明、需求可测试矩阵和开发文档确保测试设计规格说明的大纲是恰当的。如果考虑可追踪性将是比较困难的,则选择最具有综合性的文档用于软件的开发和维护(可能是软件需求规格说明或其他文档)。测试用例和规程设计应遵循此大纲,以保证需求的可追踪性。

(2) 测试工程师根据测试计划生成测试设计规格说明。

(3) 测试工程师审查从需求测试矩阵分配到测试和设计规格说明的每一测试要求,并给出测试用例的逻辑集大纲。

(4) 测试工程师根据测试计划生成测试用例规格说明。

(5) 测试工程师根据测试用例分配和可追踪的信息更新需求测试矩阵。

(6) 测试工程师审查从更新的需求测试矩阵分配到测试和设计规划说明的每一测试要求,并给出测试用例的逻辑集大纲。

(7) 测试工程师根据测试计划生成测试规程规格说明。

(8) 测试工程师根据测试规程分配和可追踪的信息更新需求测试矩阵。

(9) 测试工程师审查从需求测试矩阵分配给每个测试规程的需求,收集与每个规程相关的任何附加开发文档。除了下一步之外,这一步骤将贯穿项目此后部分。

(10) 测试工程师参考更新的需求测试矩阵分配给每个规程的需求和任何附加开发文档,给出执行软件相关所有相关测试场景的详细说明。在整个项目中随着特定功能的新信息不断产生,测试工程师将信息以需求测试矩阵中测试需求的形式收集。

(11) 测试工程师准备所有测试规程说明、测试用例规格说明和测试规程规格说明,并更新需求测试矩阵用于发布和评审。

本活动完成的标志是生成批准的测试设计规格说明。

4. 测试规程实现

测试设计规格说明、测试用例规格说明、测试规程和更新的需求测试矩阵完成后,即可进入测试规程实现阶段,该阶段活动步骤如下。

(1) 测试工程师审查需求测试矩阵、测试设计规格说明、测试用例规格说明和测试规程,为测试步骤的准备、检查、更新和发布准备详细的工作计划。

(2) 测试负责人从测试工程师获取详细的工作计划,进行计划、跟踪和监督。

(3) 测试工程师根据客户要求的深度撰写详细规程。该规程应能清晰地显示测试规程说明中的场景是如何被覆盖的。

(4) 测试工程师根据规程实现活动进程的每周报告,分发给项目经理和软件测试组。

(5) 测试负责人根据测试工程师提供的信息更新高一级的工作计划。

(6) 测试负责人准备测试规程实现活动进程的每周报告，分发给项目经理和软件测试组。

(7) 测试工程师引导进行测试规程的正式技术评审。

(8) 测试工程师基于正式技术评审更新测试规程。

(9) 测试工程师基于测试规程的活动更新需求测试矩阵。

(10) 测试工程师打印出最终的可执行规程发布给客户。

(11) 测试负责人将测试规程和需求测试矩阵发给客户。

测试规程实现活动完成的标志是生成经批准的测试规程和更新需求测试矩阵。

5. 测试执行

在完成测试规程后进入测试执行阶段，该阶段活动步骤如下。

(1) 在测试之前，测试负责人和测试工程师为即将进行的测试准备执行规程检查表。

(2) 测试工程师确保所有的规程都经过评审和更新。

(3) 测试工程师、系统工程师、开发工程师协同工作，为测试事件建立基线和实验设置。所有人都必须知道哪些内容属于基线的范围。

(4) 在测试事件开始前两周，测试工程师、软件测试组和开发工程师应执行规程中已发现的软件和测试文档中存在的问题。

(5) 测试工程师和客户或质量管理员一起执行测试。

(6) 根据测试事件生成软件问题报告。

(7) 测试工程师按照测试计划的定义准备软件测试报告。

6. 总结生成报告

在完成测试执行活动后进入总结生成报告阶段。

该活动主要任务是测试负责人根据测试计划、测试规程和软件问题报告，分析测试执行结果，总结生成软件测试报告。

活动的结束标志是生成软件测试报告。

练 习 题

1. 简述软件测试的意义。

2. 什么是软件缺陷？它的表现形式有哪些？

3. 简单分析软件缺陷产生的原因，其中哪个阶段引入的缺陷最多，修复成本又最低？

4. 当用户登录某网站购物完毕并退出后，忽然想查查购物时付账的总金额，于是按了浏览器左上角的“退回”按钮，就又回到了退出前的网页，你认为该购物软件有缺陷吗？如有，属于哪一类？

5. 什么是软件测试？简述其目的与原则。

6. 软件测试阶段是如何划分的？

7. 简述软件开发的几个模式，并说明每种模式对软件测试的影响。

8. 简述软件测试过程。

9. “软件测试能够保证软件的质量”这句话对吗？软件测试和软件质量之间是什么关系？

10. 判断以下说法是否正确。

(1) 软件测试和软件调试是同一回事。

(2) 软件测试是可以穷尽的。

(3) 测试是为了证明软件的正确性。

(4) 测试过程中应重视测试的执行,可以轻视测试的设计。

(5) 测试不能修复所有的软件故障。

(6) 因为测试工作简单,对软件产品影响不大,所以可以把测试作为新员工的一个过渡工作,或安排不合格的开发人员做测试。

11. 简述软件开发进程与测试进程的关系。

第 2 章　软件测试方法与过程

2.1　软件测试复杂性与经济性

人们常常以为，开发一个程序是困难的，测试一个程序则比较容易，然而事实并非如此。在软件测试当中，由于各种原因，不能实现对软件进行完全的测试并找出所有的软件缺陷，使软件达到完美无缺的理想状态。设计测试用例是一项细致并需要高度技巧的工作，稍有不慎就会顾此失彼，发生不应有的疏漏。除此之外，要通过测试找出软件中的所有故障也是不现实、不可能的，这涉及软件测试的复杂性、充分性和经济性。

1. 软件测试的复杂性

（1）无法对程序进行完全的测试

不论是黑盒测试方法还是白盒测试方法，由于测试情况数量巨大，都不可能进行彻底的测试。所谓彻底测试，就是让被测程序在一切可能的输入情况下全部执行一遍。通常也称这种测试为“穷举测试”。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。“白盒”法是穷举路径测试，贯穿程序的独立路径数是天文数字，要使每条路径都得到测试是不现实的。

软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成测试。为了降低测试成本，选择测试用例时应注意遵守“经济性”的原则。第一，要根据程序的重要性和一旦发生故障将造成的损失来确定它的测试等级；第二，要认真研究测试策略，以便能使用尽可能少的测试用例，发现尽可能多的程序错误。掌握好测试量是至关重要的，一位有经验的软件开发管理人员在谈到软件测试时曾这样说过：“不充分的测试是愚蠢的，而过度的测试是一种罪孽”。测试不足意味着让用户承担隐藏错误带来的危险，过度测试则会浪费许多宝贵的资源。

（2）测试无法保证被测程序中无遗留错误

软件测试工作与传染病疫情员的工作是很相似的，疫情员只是报告已经发现的疫情，却无法报告潜伏的疫情状况。同样通过软件测试只能报告软件已经被发现的缺陷和故障，但不能保证经测试后发现的是全部的软件缺陷，即无法报告隐藏的软件故障。若能继续进行测试工作，可能会发现一些新的问题。在实际测试中，穷举测试工作量太大，实践上行不通，这就注定了一切实际测试都是不彻底的，当然就不能够保证被测试程序中不存在遗留的错误。而且在“白盒”法中即使实施了穷举路径测试，程序仍然可能有错误。第一，穷举路径测试决不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可

能查出程序中因遗漏路径而出错。第三，穷举路径测试可能发现不了一些与数据相关的错误。E. W. Dijkstra 的一句名言对此作了很好的注解："程序测试只能证明错误的存在，但不能证明错误不存在"。

(3) 不能修复所有的软件故障

在软件测试中，严峻的现实是：即使付出再多的时间和代价，也不能够使所有的软件故障都得到修复。但这并不说明测试没有达到目的，关键是要进行正确的判断、合理的取舍，根据风险分析决定哪些故障必须修复，哪些故障可以不修复。通常不能修复软件故障的理由如下。

① 没有足够的时间进行修复。

② 修复的风险较大。修复了旧的故障可能产生更多的故障。

③ 不值得修复。主要是不常用的功能中的故障，或对运行影响不大的故障。

④ 可不算做故障的一些缺陷。在某些场合，错误理解或者软件规格说明变更可以将软件故障当作附加的功能而不作为故障来对待。

2. 软件测试的经济性

如果不能做到测试软件所有的情况，则该软件就是有风险的。软件测试不可能对软件使用中所有的情况进行测试，但有可能客户会在使用软件的时候遇到，并且可能发现软件的缺陷。等到这个时候再进行软件缺陷的修复，代价是很高的。

软件测试的一个主要工作原则就是如何将无边无际的可能性减小到一个可以控制的范围，以及如何针对软件风险做出恰当的选择，去粗存精，找到最佳的测试量，使得测试工作量不多不少，既能达到测试的目的，又能较为经济。

测试是软件生存期中费用消耗最大的环节。测试费用除了测试的直接消耗外，还包括其他的相关费用。决定需要做多少次测试的主要影响因素如下。

(1) 系统的目的

系统的目的差别在很大程度上影响所需要进行的测试的数量。那些可能产生严重后果的系统必须要进行更多的测试。一台在 Boeing 757 上的系统应该比一个用于公共图书馆中检索资料的系统需要更多的测试。一个用来控制密封燃气管道的系统应该比一个与有毒爆炸物品无关的系统有更高的可信度。一个安全关键软件的开发组比一个游戏软件开发组要有苛刻得多的查找错误方面的要求。

(2) 潜在的用户数量

一个系统的潜在用户数量也在很大程度上影响了测试必要性的程度。这主要是由于用户团体在经济方面的影响。一个在全世界范围内有几千个用户的系统肯定比一个只在办公室中运行的有两三个用户的系统需要更多的测试。如果不能使用的话，前一个系统的经济影响肯定比后一个系统大。除此以外，在分配处理错误的时候，所花的代价的差别也很大。如果在内部系统中发现了一个严重的错误，在处理错误时所花费用就相对少一些，如果要处理一个遍布全世界的错误就需要花费相当大的财力和精力。

(3) 信息的价值

在考虑测试的必要性时，还需要将系统中所包含的信息的价值考虑在内，一个支持许多家大银行或众多证券交易所的客户机/服务器系统中含有经济价值非常高的内容。很显然这一系统需要比一个支持鞋店的系统要进行更多的测试。这两个系统的用户都希望得到高

质量、无错误的系统,但是前一种系统的影响比后一种要大得多。因此我们应该从经济方面考虑,投入与经济价值相对应的时间和金钱去进行测试。

(4) 开发机构

一个没有标准和缺少经验的开发机构很可能开发出充满错误的系统。在一个建立了标准和有很多经验的开发机构中开发出来的系统,错误不会很多,因此,对于不同的开发机构来说,所需要的测试的必要性也就截然不同。然而,那些需要进行大幅度改善的机构反而不大可能认识到自身的弱点,那些需要更加严格的测试过程的机构往往是最不可能进行这一活动的,在许多情况下,机构的管理部门并不能真正地理解开发一个高质量的系统的好处。

(5) 测试的时机

测试量会随时间的推移发生改变。在一个竞争很激烈的市场里,争取时间可能是制胜的关键,开始可能不会在测试上花多少时间,但几年后如果市场分配格局已经建立起来了,那么产品的质量就变得更重要了,测试量就要加大。测试量应该针对合适的目标进行调整。

2.2 软件测试方法

软件测试的策略、方法和技术是多种多样的。对于软件测试方法,可以从不同的角度得到以下基本分类: 从是否需要执行被测软件的角度,可分为静态测试和动态测试; 从测试是否针对系统的内部结构和具体实现算法的角度来看,可分为白盒测试和黑盒测试; 根据执行测试的主体不同,又可以将测试方法分为人工测试和自动化测试。

2.2.1 静态测试与动态测试

1. 静态测试

在软件开发过程中,每产生一个文档,都必须对它进行测试,以检测它的质量是否满足要求。这样的检查工作与全面质量管理的思想是一致的,也与项目管理过程相一致。每当一个文档通过了静态测试,就标志着一项开发工作的总结,标志着项目取得了一定的进展,进入了一个新的阶段。

静态测试的基本特征是在对软件进行分析、检查和测试时不实际运行被测试的程序。它可以用于对各种软件文档进行测试,是软件开发中十分有效的质量控制方法之一。在软件开发过程中的早期阶段,由于可运行的代码尚未产生,不可能进行动态测试,而这些阶段的中间产品的质量直接关系到软件开发的成败与开销的大小,因此,在这些阶段,静态测试的作用尤为重要。在软件开发多年的生产实践经验和教训的基础上,人们总结出了一些行之有效的静态测试技术和方法,如结构化走通、正规检视等。这些方法和测试技术可以与软件质量的定量度量技术相结合,对软件开发过程进行监视、控制,从而保障软件质量。

针对程序代码的静态测试是指不运行被测程序本身,仅通过分析或检查源程序的文法、结构、过程、接口等来检查程序的正确性。静态方法通过程序静态特性的分析,找出欠缺和可疑之处,例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错,并为测试用例选取提供指导。

针对代码的静态测试包括代码检查、静态结构分析、代码质量度量等。它可以由人工进

行，充分发挥人的逻辑思维优势，也可以借助软件工具自动进行。

(1) 代码检查

代码检查主要检查代码和设计的一致性，代码对标准的遵循及可读性，代码的逻辑表达的正确性，代码结构的合理性等方面；可以发现违背程序编写标准的问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

在实际使用中，代码检查比动态测试更有效率，能快速找到缺陷，发现 30%～70%的逻辑设计和编码缺陷。代码检查看到的是问题本身而非征兆。但是代码检查非常耗费时间，而且代码检查需要知识和经验的积累。代码检查应在编译和动态测试之前进行，在检查前，应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表等。

(2) 静态结构分析

静态结构分析主要是以图形的方式表现程序的内部结构，例如函数调用关系图、函数内部控制流图。其中，函数调用关系图以直观的图形方式描述一个应用程序中各个函数的调用和被调用关系；控制流图显示一个函数的逻辑结构，它由许多结点组成，一个结点代表一条语句或数条语句，连接结点的叫边，边表示结点间的控制流向。

(3) 代码质量度量

ISO/IEC 9126 国际标准所定义的软件质量包括 6 个方面：功能性、可靠性、易用性、效率、可维护性和可移植性。软件的质量是软件属性的各种标准度量的组合。

针对软件的可维护性，目前业界主要存在三种度量参数：Line 复杂度、Halstead 复杂度和 McCabe 复杂度。其中 Line 复杂度以代码的行数作为计算的基准。Halstead 以程序中使用到的运算符与运算元数量作为计数目标(直接测量指标)，然后可以据此计算出程序容量、工作量等。McCabe 复杂度一般称为圈复杂度(Cyclomatic complexity)，它将软件的流程图转化为有向图，然后以图论来衡量软件的质量。McCabe 复杂度包括圈复杂度、基本复杂度、模块设计复杂度、设计复杂度和集成复杂度。

2. 动态测试

所谓动态测试是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能，动态测试包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

(1) 功能确认与接口测试。这部分的测试包括各个单元功能的正确执行、单元间的接口，包括单元接口、局部数据结构、重要的执行路径、错误处理的路径和影响上述几点的边界条件等内容。

(2) 覆盖率分析。覆盖率分析主要对代码的执行路径覆盖范围进行评估，语句覆盖、判定覆盖、条件覆盖、条件/判定覆盖、修正条件/判定覆盖、基本路径覆盖都是从不同要求出发，为设计测试用例提供依据的。

(3) 性能分析。代码运行缓慢是开发过程中一个重要问题。一个应用程序运行速度较慢，程序员不容易找到哪里出现了问题。如果不能解决应用程序的性能问题，将降低并极大地影响应用程序的质量，于是查找和修改性能瓶颈成为调整整个代码性能的关键。目前性能分析工具大致分为纯软件的测试工具、纯硬件的测试工具(如逻辑分析仪和仿真器等)和

软硬件结合的测试工具三类。

(4) 内存分析。内存泄漏会导致系统运行的崩溃,尤其对于嵌入式系统这种资源比较匮乏、应用非常广泛,而且往往又处于重要部位的,将可能导致无法预料的重大损失。通过测量内存使用情况,我们可以了解程序内存分配的真实情况,发现对内存的不正常使用,在问题出现前发现征兆,在系统崩溃前发现内存泄露错误,发现内存分配错误,并精确显示发生错误时的上下文情况,指出发生错误的原由。

2.2.2 黑盒测试与白盒测试

1. 黑盒测试

黑盒测试是指在对程序进行的功能抽象的基础上,将程序划分成功能单元,然后对每个功能单元生成的测试数据进行测试。黑盒测试也称功能测试或数据驱动测试,它是已知产品所应具有的功能,通过测试来检测每个功能是否都能正常使用。在测试时,把程序看作一个不能打开的黑盒子,在完全不考虑程序内部结构和内部特性的情况下,测试者在程序接口进行测试,只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能适当接收输入数据而产生正确的输出信息,并且保持外部信息的完整性。

在黑盒测试中,被测软件的输入域和输出域往往是无限域,因此穷举测试通常是不可行的,必须以某种策略分析软件规格说明,从而得出测试用例集,尽可能全面而又高效地对软件进行测试。下面就几种功能测试的方法进行简单介绍,具体说明会在后面的章节进行。

(1) 等价类划分

所谓等价类,就是某个输入域的集合,集合中的每个输入对揭露程序错误来说是等效的,把程序的输入域划分成若干部分,然后从每个部分中选取少数代表性数据作为测试用例,这就是等价类划分方法。它是功能测试的基本方法。

(2) 因果图法

因果图是一种形式语言,由自然语言写成的规范转换而成,这种形式语言实际上是一种使用简化记号表示数字逻辑图。因果图法是帮助人们系统地选择一组高效测试用例的方法,此外,它还能指出程序规范中的不完全性和二义性。

(3) 边界值分析

实践证明,软件在输入、输出域的边界附近容易出现差错,边界值分析是考虑边界条件而选取测试用例的一种功能测试方法。所谓边界条件,是相对于输入和输出等价类直接在其边界上,或稍高于和稍低于其边界的这些状态条件。边界值分析是对等价类划分的有效补充。

另外常见的黑盒测试方法还有基于决策表的测试、错误推测法等。

2. 白盒测试

白盒测试是根据被测程序的内部结构设计测试用例的一类测试,又称为结构测试或逻辑驱动测试,它是知道产品内部工作过程,通过测试来检测产品内部动作是否按照规格说明书的规定正常进行,按照程序内部的结构测试程序,检验程序中的每条通路是否都能按预定要求正确工作。白盒法全面了解程序内部逻辑结构、对所有逻辑路径进行测试,是穷举路径测试。在使用这一方案时,测试者必须检查程序的内部结构,从检查程序的逻辑着手,得出测试数据,测试程序的内部变量状态、逻辑结构、运行路径等,检验程序中的每条通路是否都

能按预定要求正确工作,所有内部成分是否按规定正常进行。

贯穿程序的所有路径数是天文数字,所以白盒测试法在实际操作时不可能实现路径的穷举,它常以达到对程序内部结构的某种覆盖标准为目标。白盒测试主要用于软件验证,其主要方法有逻辑覆盖、基本路径测试和数据流测试等。

不同的测试方法各有所长,都能比较容易地发现某种类型的错误,却不易发现其他类型的错误,各有侧重、各有优缺点,构成互补关系。白盒测试可以有效地发现程序内部的编码和逻辑错误,但无法检验出程序是否完成了规定的功能;黑盒测试可以根据程序的规格说明检测出程序是否完成了规定的功能,但未必能够提供对代码的完全覆盖,而且规格说明往往会出现歧义或不完整的情况,这在一定程度上降低了黑盒测试的效果。因此在实际测试中,应结合各种测试方法形成综合策略。一般在单元测试阶段主要用白盒测试,在系统测试时主要用黑盒测试。

实际上,黑盒测试法和白盒测试法的界限现在已经变得越来越模糊了,单纯地根据规约或代码生成测试用例都不是很现实。目前已有越来越多的人在尝试将这两种方法结合起来,例如根据规格说明来生成测试用例,然后根据代码(静态分析或动态执行代码)来进行测试用例的取舍和精化等,以至形成了所谓的“灰盒测试”法。这也是目前软件测试的一个发展方向。

2.2.3 人工测试与自动化测试

1. 人工测试

广义上,人工测试是人为测试和手工测试的统称。人为测试的主要方法有桌前检查(desk checking),代码审查(code review)和走查(walkthrough)。事实上,用于软件开发各个阶段的审查(inspection)或评审(review)也是人为测试的一种。经验表明,使用这种方法能够有效地发现30%到70%的逻辑设计和编码错误。由于人为测试技术在检查某些编码错误时,有着特殊的功效,它常常能够找出利用计算机不容易发现的错误。人为测试至今仍是一种行之有效的测试方法。手工测试指的是在测试过程中,按测试计划一步一步执行程序,得出测试结果并进行分析的测试行为。目前,在功能测试中经常使用这种测试方法。

2. 自动化测试

自动化测试指的是利用测试工具来执行测试,并进行测试结果分析的测试行为。自动化测试不可能完全自动,它离不开人的智力劳动。但是它能替代人做一些烦琐或不可能通过手工达到的事情。由于测试工作的繁重性、重复性等特征,自动化测试是提高测试效率的一个有效方法,也是目前测试研究领域的一个热点。

2.3 软件测试阶段

软件测试贯串软件产品开发的整个生命周期,软件项目一开始软件测试也就开始了。从过程来看,软件测试是由一系列的不同测试阶段所组成的。这些阶段分为规格说明书审查、系统和程序设计审查、单元测试、集成测试、确认测试、系统测试以及验收(用户)测试。软件开发的过程是自顶向下的,测试则正好相反,上述过程就是自底向上、逐步集成的。

1. 规格说明书审查

为保证需求定义的质量，应对需求分析规格说明书进行严格的审查。由测试人员参与系统或产品需求分析，认真阅读有关用户需求分析文档，真正理解客户的需求，检查规格说明书对产品描述的准确性、一致性等，为今后熟悉应用系统、编写测试计划、设计测试用例等做好准备工作。

2. 系统和程序设计审查

代码会审是一种静态的白盒测试方法，是由一组人通过阅读、讨论来审查程序结构、代码风格、算法等的过程。会审小组由组长、3～5名程序设计人员、编程人员和测试人员组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开代码会审会。实践表明，代码会审做得好的话可以发现大部分程序缺陷，甚至程序员在自己讲解过程中就能发现不少代码错误，而讨论可能进一步促使问题暴露。

3. 单元测试

单元测试集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。

4. 集成测试

该阶段把已测试过的模块组装起来，主要对与设计相关的软件体系结构的构造进行测试。

5. 确认测试

检查已实现的软件是否满足了需求规格说明中确定了的各种需求以及软件配置是否完全、正确。

6. 系统测试

把已经经过确认的软件纳入实际运行环境中，与其他系统成分组合在一起进行测试。

7. 验收测试

检验软件产品的最后一道工序，主要突出用户的作用，同时软件开发人员也应在一定的程度上参与。

2.4 单元测试

单元测试又称模块测试，是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。这个阶段更多关注程序实现的细节，需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

2.4.1 单元测试主要任务

在单元测试时，测试者需要依据详细设计说明书和源程序清单，了解该模块的I/O条件和模块的逻辑结构，主要采用白盒测试的测试用例，辅之以黑盒测试的测试用例，使之对任何合理的输入和不合理的输入都能鉴别和响应。它主要测试以下几方面的问题。

1. 模块接口测试

(1) 单元测试的开始，应对通过被测模块的数据流进行测试。测试项目包括以下内容。

① 调用本模块的输入参数是否正确；
② 本模块调用子模块时输入给子模块的参数是否正确；
③ 全局量的定义在各模块中是否一致；
④ 是否修改了只做输入用的形式参数。
(2) 在做内外存交换时需要考虑以下内容。
① 文件属性是否正确；
② OPEN 与 CLOSE 语句是否正确；
③ 缓冲区容量与记录长度是否匹配；
④ 在进行读写操作之前是否打开了文件；
⑤ 在结束文件处理时是否关闭了文件；
⑥ 正文书写/输入错误；
⑦ I/O 错误是否检查并做了处理。

2. 局部数据结构测试

(1) 不正确或不一致的数据类型说明。
(2) 使用尚未赋值或尚未初始化的变量。
(3) 错误的初始值或错误的默认值。
(4) 变量名拼写错误或书写错误。
(5) 不一致的数据类型。
(6) 上溢、下溢或地址异常。

3. 路径测试

(1) 选择适当的测试用例，对模块中重要的执行路径进行测试。
(2) 应当设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。
(3) 对基本执行路径和循环进行测试可以发现大量的路径错误。

4. 错误处理测试

(1) 出错的描述是否难以理解。
(2) 出错的描述是否能够对错误定位。
(3) 显示的错误与实际的错误是否相符。
(4) 对错误条件的处理是否正确。
(5) 在对错误进行处理之前，错误条件是否已经引起系统的干预等。

5. 边界测试

注意数据流、控制流中刚好等于、大于或小于确定的边界值时出错的可能性，对这些地方要仔细地选择测试用例，认真加以测试。

如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。

2.4.2 单元测试执行过程

通常单元测试在编码阶段进行，在源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功

能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。这些辅助模块分为以下两种。

驱动模块(driver)：用以模拟被测模块的上级模块，它接收测试数据，把这些数据传送给被测模块，启动被测模块，最后输出实测结果。

桩模块(stub)：也称为存根程序，用以模拟被测模块工作过程中所调用的子模块。桩模块由被测模块调用，它们一般只进行很少的数据处理(例如打印入口和返回)，以便于检验被测模块与其下级模块的接口。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做。

被测模块与它相关的驱动模块及桩模块共同构成了一个"测试环境"。如果一个模块要完成多种功能，且以程序包或对象类的形式出现(例如 Ada 中的包、MODULA 中的模块、C++ 中的类)，这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互联测试。有人把这种情况特别称为模块测试，以区别单元测试。

2.5 集成测试

集成测试，也称为组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求组装成为子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。

2.5.1 集成模式

选择什么样的方式把模块组装起来形成一个可运行的系统，直接影响到测试成本、测试计划、测试用例的设计、测试工具的选择等。通常有两种集成方式：一次性集成方式和增量式集成方式。

1. 一次性集成测试模式

它是一种非增量式组装方式，也叫做整体拼装。使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。

2. 增量式集成测试模式

增量式的测试方法与非增量式的测试不同，它的集成是逐步实现的，集成测试也是逐步完成的，又称渐增式集成，也可以说它将单元测试与集成测试结合起来进行。首先对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在集成的过程中边连接边测试，以发现连接过程中产生的问题，通过增值逐步组装成为要求的软件系统。

一次性集成测试的方法是先分散测试，然后集中在一起再一次完成集成测试。假如在模块的接口处存在错误，只会在最后的集成测试时一下子暴露出来。这时为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，新旧错误混杂，更难断定出错的原因和位置。与此相反，增量式集成测试的逐步集成和逐步测试的方法，将可能出现的差错分散暴露出来，错误易于定位和纠正。而且一些模块在逐步集成的测试中，得到了

较多次的考验，因此，接口测试更加彻底，能取得较好的测试结果。总之，增量式测试要比非增量式测试具有一定的优越性。两种模式中，增量式测试模式需要编写的 Driver 或 Stub 程序较多、发现模块间接口错误相对稍晚，但增量式测试模式还是具有明显的优势。一般不推荐使用一次性集成测试模式，不过在规模较小的应用系统中较适用。

2.5.2 集成方法

当对两个以上模块进行集成时，不可能忽视它们和周围模块的相互联系。为模拟这种联系，需设置若干辅助测试模块，也就是连接被测试模块的程序段。和单元测试阶段一样，辅助模块通常有驱动模块和桩模块两种。

增量式集成测试可以按照不同的次序实施，因此通常有三种不同的方法，自顶向下集成、自底向上集成和混合集成。

1. 自顶向下集成

自顶向下集成是从主控模块开始，按照软件的控制层次结构向下逐步把各个模块集成在一起。集成过程中可以采用深度优先或广度优先的策略。其中按深度方向组装的方式，可以首先实现和验证一个完整的软件功能。

自顶向下集成的具体步骤如下。

(1) 对主控模块进行测试，测试时用桩程序代替所有直接附属于主控模块的模块；

(2) 根据选定的结合策略（深度优先或广度优先），每次用一个实际模块代替一个桩模块（新结合进来的模块往往又需要新的桩模块）；

(3) 在结合下一个模块的同时进行测试；

(4) 为了保证加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。

从第(2)步开始不断地重复进行上述过程，直至完成。

自顶向下集成能尽早地对程序的主要控制和决策机制进行检验，因此能较早地发现错误。但是在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，因此测试并不充分。自顶向下集成不需要驱动模块，但需要建立桩模块，要使桩模块能够模拟实际子模块的功能十分困难，因为桩模块在接收了所测模块发送的信息后需要按照它所代替的实际子模块功能返回应该回送的信息，这必将增加建立桩模块的复杂度，而且导致增加一些附加的测试。另外，涉及复杂算法和真正输入输出的模块一般在底层，它们是最容易出问题的模块，到组装和测试的后期才遇到这些模块，一旦发现问题，会导致过多的回归测试。

2. 自底向上集成

自底向上集成是从“原子”模块（即软件结构最低层的模块）开始组装测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）已经组装并测试完成，所以不再需要桩模块，在模块的测试过程中需要从子模块中得到的信息可以直接运行子模块得到。其具体步骤如下。

(1) 把低层模块组合成实现某个特定软件子功能的族；

(2) 写一个驱动程序（用于测试的控制程序），协调测试数据的输入和输出；

(3) 对由模块组成的子功能族进行测试；

(4) 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族。

从第(2)步开始不断重复进行上述过程，直至完成。

自底向上集成的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”。就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。但这种方式的优点是不需要桩模块，而建立驱动模块一般比建立桩模块容易，同时由于涉及复杂算法和真正输入输出的模块最先得到组装和测试，可以把最容易出问题的部分在早期解决。此外自底向上集成可以实施多个模块的并行测试，提高测试效率。

3. 混合集成

自顶向下增值的方式和自底向上增值的方式各有优缺点。一般来讲，一种方式的优点是另一种方式的缺点，具体测试时通常是把以上两种方式结合起来进行集成和测试。混合集成是自顶向下和自底向上集成的组合。一般对软件结构的上层使用自顶向下结合的方法，对下层使用自底向上结合的方法。

另外在组装测试时，应当确定关键模块，并尽量对这些关键模块及早进行测试。关键模块的特征是：满足某些软件需求；在程序的模块结构中位于较高的层次(高层控制模块)；较复杂、较易发生错误；有明确定义的性能要求。

2.5.3 持续集成

在实际测试中，应该将不同集成模式有机结合起来，采用并行的自顶向下、自底向上混合集成方式，而更重要的是采取持续集成的策略。软件开发中各个模块不是同时完成，根据进度将完成的模块尽可能早的进行集成，有助于尽早发现缺陷，避免集成阶段大量缺陷涌现。同时自底向上集成时，先期完成的模块将是后期模块的桩程序，而自顶向下集成时，先期完成的模块将是后期模块的驱动程序，从而使后期模块的单元测试和集成测试出现了部分的交叉，这样不仅节省了测试代码的编写，也提高了工作效率。

如果不采用持续集成策略，开发人员经常需要集中分析软件究竟在什么地方出了错。因为某个程序员在写自己这个模块代码时，可能会影响其他模块的代码，造成与已有程序的变量冲突、接口错误，结果导致缺陷的产生。随着时间的推移，问题会逐渐恶化，通常，在集成阶段出现的缺陷早在几周甚至几个月之前就已经存在了。结果，开发者需要在集成阶段耗费大量的时间和精力来寻找这些缺陷的根源。如果使用持续集成，这样的缺陷绝大多数都可以在引入的第一天就被发现。而且，由于一天之中发生变动的部分并不多，所以可以很快找到出错的位置，这也是为什么进行每日构建软件包的原因所在。所以，持续集成可以减少集成阶段消灭缺陷所消耗的时间，从而提高软件开发的质量与效率。

2.5.4 回归测试

在软件生命周期中的任何一个阶段，只要软件发生了改变，就可能给该软件带来问题。软件的改变可能是源于发现了错误并做了修改，也有可能是因为在集成或维护阶段加入了新的模块。在增量型软件开发过程中，通常将软件分阶段进行开发，在一个阶段的软件开发结束后，将被测软件交给测试组进行测试，而下一个阶段增加的软件又有可能对原来的系统造成破坏。因此，每当软件发生变化时，我们就必须进行回归测试，重新测试原有的功能，以

便确定修改是否达到了预期的目的,检查修改是否损害了原有的正常功能。

具体的方法:对修改过的代码重新运行现有的测试,确定更改是否破坏了在更改之前有效的任何事物,并且在必要的地方编写新测试。执行回归测试时,首要考虑的应该是覆盖范围足够大但不浪费时间。尽可能少花时间执行回归测试,但不减少在旧的、已经测试过的代码中检测新失败的可能性。

此过程中需考虑的一些策略和因素包括下列内容。

(1) 测试已修复的错误。程序员可能已经处理了症状,但并未触及根本原因。

(2) 监视修复的副作用。错误本身可能得到了修复,但修复也可能造成其他错误。

(3) 为每个修复的错误编写一个回归测试。

(4) 如果两个或更多的测试类似,确定哪一个效率较低并将其删除。

(5) 识别程序始终通过的测试并将它们存档。

(6) 集中考虑功能性问题,而不是与设计相关的问题。

(7) 更改数据(更改量可多可少)并找出任何产生的损坏。

(8) 跟踪程序内存更改的效果。

在实际工作中,回归测试需要反复进行,当测试者一次又一次地完成相同的测试时,这些回归测试将变得非常令人厌烦,而在大多数回归测试需要手工完成的时候尤其如此,因此,需要通过自动化测试来实现重复的和一致的回归测试。通过测试自动化可以提高回归测试效率。在测试软件时,应用多种测试技术是常见的。当测试一个修改了的软件时,测试者也可能希望采用多于一种回归测试策略来增加对修改软件的信心。不同的测试者可能会依据自己的经验和判断选择不同的回归测试技术和策略。因此为了支持多种回归测试策略,自动化测试工具应该是通用的和灵活的,以便满足达到不同回归测试目标的要求。

回归测试并不减少对系统新功能和特征的测试需求,回归测试也包括新功能和特征的测试。如果回归测试包不能达到所需的覆盖要求,必须补充新的测试用例使覆盖率达到规定的要求。

回归测试是重复性较多的活动,容易使测试者感到疲劳和厌倦,降低测试效率,在实际工作中可以采用一些策略减轻这些问题。例如,安排新的测试者完成手工回归测试,分配更有经验的测试者开发新的测试用例,编写和调试自动化测试脚本,做一些探索性的测试。还可以在不影响测试目标的情况下,鼓励测试者创造性地执行测试用例,变化的输入、按键和配置有助于激励测试者揭示出新的错误。

在组织回归测试时需要注意两点,首先是各测试阶段发生的修改一定要在本测试阶段内完成回归,以免将错误遗留到下一测试阶段。其次,回归测试期间应对该软件版本冻结,将回归测试发现的问题集中修改,集中回归。

在实际工作中,可以将回归测试与兼容性测试结合起来进行。在新的配置条件下运行旧的测试可以发现兼容性问题,同时也可以揭示编码在回归方面的错误。

2.6 确认测试

确认测试又称有效性测试。它的任务是验证软件的有效性,即验证软件的功能和性能及其他特性是否与用户的要求一致。在软件需求规格说明书描述了全部用户可见的软件属

性，其中有一节叫做有效性准则，它包含的信息就是软件确认测试的基础。

在确认测试阶段主要进行有效性测试以及软件配置复审。

1. 进行有效性测试（功能测试）

有效性测试是在模拟的环境（可能就是开发的环境）下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。为此，需要首先制定测试计划，规定要做测试的种类。还需要制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确的且便于使用。同时，对其他软件需求，例如可移植性、兼容性、出错自动恢复、可维护性等，也都要进行测试，确认是否满足。

2. 软件配置复查

软件配置复查的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，具有维护阶段所必需的细节，而且已经编排好分类的目录。

除了按合同规定的内容和要求，由人工审查软件配置之外，在确认测试的过程中，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误，并且适当地补充和改正。

2.7 系统测试

所谓系统测试，是将通过确认测试的软件，作为基于整个计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的严格有效的测试以发现软件的潜在问题，保证系统的运行。

系统测试明显区别于功能测试。功能测试主要是验证软件功能的实现情况，不考虑各种环境以及非功能问题，如安全性、可靠性、性能等，而系统测试是在更大的范围内进行的测试，着重对系统的性能、特性进行测试。它的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。所以系统测试的测试用例应该根据需求分析规格说明来设计，并在实际使用环境下来运行。

下面对系统测试的内容进行简要介绍。

1. 强度测试

强度测试是要检查在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试。强度测试需要在反常规数据量、频率或资源的方式下运行系统，以检查系统能力的最高实际限度。例如，输入数据速率提高一个数量级，确定输入功能将如何响应；或设计需要占用最大存储量或用其他资源的测试用例进行测试。

强度测试的一个变种就是敏感性测试。在程序有效数据界限内的一个小范围内的一组数据可能引起极端的或不平稳的错误处理出现，或者导致极度的性能下降的情况发生。此测试用以发现可能引起这种不稳定性或不正常处理的某些数据组合。

2. 性能测试

性能测试用来测试软件在系统集成中的运行性能，检查其是否满足需求说明书中规定

的性能，特别是对于实时系统或嵌入式系统，仅提供符合功能需求但不符合性能需求的软件是不能接受的。性能测试可以在测试过程的任意阶段进行，即使是在单元层，但只有当整个系统的所有成分都集成在一起后，才能检查一个系统的真正性能。性能测试常常需要与强度测试结合起来进行，并常常要求同时进行硬件和软件检测，这就是说，常常有必要在一种苛刻的资源环境中衡量资源的使用。通常，对软件性能的检测表现在以下几个方面：响应时间、吞吐量、辅助存储区（例如缓冲区、工作区的大小等）、处理精度等。

外部的测试设备可以检测测试的执行，当出现某种情况时可以记录下来。通过对系统的检测，测试者可以发现导致效率降低和系统故障的原因。为了记录性能，需要在系统中安装必要的量测仪表或者为度量性能而设置的软件。

3. 恢复测试

恢复测试是要证实在克服硬件故障（包括掉电、硬件或网络出错等）后，系统能正常地继续进行工作，并不对系统造成任何损害。为此，可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错，并由此检查系统的错误探测功能——系统能否发现硬件失效与故障；能否切换或启动备用的硬件；在故障发生时能否保护正在运行的作业和系统状态；在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业等。例如掉电测试，它的目的是测试软件系统在发生电源中断时能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

4. 安全测试

任何管理敏感信息或者能够对个人造成不正当伤害的计算机系统都是不正当或非法侵入的目标。通常力图破坏系统的保护机构以进入系统的主要方法有：正面攻击或从侧面、背面攻击系统中易受损坏的那些部分；以系统输入为突破口，利用输入的容错性进行正面攻击；申请和占用过多的资源压垮系统，以破坏安全措施，从而进入系统；故意使系统出错，利用系统恢复的过程，窃取用户口令及其他有用的信息；通过浏览残留在计算机各种资源中的垃圾（无用信息），以获取如口令、安全码、译码关键字等信息；浏览全局数据，期望从中找到进入系统的关键字；浏览那些逻辑上不存在，但物理上还存在的各种记录和资料等。

安全性测试是要检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞，以检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种方法试图突破防线。系统安全设计的准则是使非法侵入的代价超过被保护信息的价值。

5. 可靠性测试

软件可靠性是软件系统在规定的时间内和规定的环境条件下，完成规定功能的能力。它是软件系统的固有特性之一，表明了一个软件系统按照用户的要求和设计目标，执行其功能的可靠程度。软件可靠性与软件缺陷有关，也与系统输入和系统使用有关。理论上说，可靠的软件系统应该是正确的、完整的、一致的和健壮的。但是实际上任何软件都不可能达到百分之百的正确，而且也无法精确度量。一般情况下，只能通过对软件系统进行测试来度量其可靠性。

可靠性测试是从验证的角度出发，为了检验系统的可靠性是否达到预期目标而进行的测试。它通过测试发现并纠正影响可靠性的缺陷，实现软件可靠性增长，并验证其是否达到

了用户的可靠性要求。该测试需要从用户的角度出发，模拟用户实际使用系统的情况，设计出系统的可操作视图。在这个基础上，根据输入空间的属性及依赖关系导出测试用例，然后在仿真的环境或真实的环境下执行测试用例并记录测试的数据。

根据在测试过程中收集获得的失效数据，如失效间隔时间、失效修复时间、失效数量、失效级别等，应用可靠性模型，可以得到系统的失效率及可靠性增长趋势。其中可靠性增长趋势是测试开始时的失效率与测试结束时的失效率之比。

从黑盒(占主要地位)和白盒测试两个角度出发有以下几种常用的可靠性模型。

(1) 黑盒方面的可靠性模型包括了基本执行时间模型(Musa)、故障分离模型(Jelinski-Moranda)、NHPP 模型及增强的 NHPP 模型(Goel-Okumoto)以及贝叶斯判定模型(Littlewood-Verrall)。

(2) 在白盒方面的可靠性模型包括了基于路径的模型和基于状态的模型。

对于相同的数据，不同的模型可以得到不同的结果，有些结果可能大相径庭，这往往是因为不同的模型基于的假设条件不同而造成的。

业界流行的可靠性模型还有很多种，不同的可靠性模型其依赖的假设条件也是不同的，使用范围也不同。因此对于一个产品，其所适合使用的可靠性模型需要根据实际出发，尽可能选择与可靠性模型假设条件相近的模型。

6. 安装测试

理想情况下，一个软件的安装程序应当平滑地集成用户的新软件到已有的系统中去，就像一个客人被介绍到一个聚会中去一样，彼此交换适当的问候。一些对话窗口提供简单的、容易理解的安装选项和支持信息，并且完成安装过程。然而，在某些糟糕的情况下，安装程序可能会出错，使新的程序无法工作，已有的功能受到影响，甚至安装过程严重损坏用户系统。

在安装软件系统时，会有多种选择：要分配和装入文件与程序库；布置适用的硬件配置；进行程序的联结。而安装测试就是要找出在这些安装过程中出现的错误，其目的是要验证成功安装系统的能力。它通常是开发人员的最后一个活动，并且通常在开发期间不太受关注。但是，它是客户使用新系统时执行的第一个操作，因此，清晰并且简单的安装过程是系统文档中最重要的部分。

7. 容量测试

容量测试是根据预先分析出反映软件系统应用特征的某项指标极限值(如最大并发用户数，最大数据库记录数等)，测试系统在其极限值状态下是否能保持主要功能正常运行。例如：对于编译程序，让它处理特别长的源程序；对于操作系统，让它的作业队列“满员”；对于信息检索系统，让它使用频率达到最大。在使用系统的全部资源达到“满负荷”的情形下，测试系统的承受能力。

容量测试的完成标准可以定义为：所计划的测试已全部执行，而且达到或超出指定的系统限制时没有出现任何软件故障。

8. 文档测试

文档测试是检查用户文档(如用户手册)的清晰性和精确性。在用户文档中所使用的例子必须在测试中测试过，确保叙述正确无误。

2.8 验收测试

验收测试是软件产品完成系统测试后，在发布之前所进行的软件测试活动，它是技术测试的最后一个阶段。通过验收测试，产品就会进入发布阶段。验收测试的目的是确保软件准备就绪，并且可以让最终用户将其用于执行软件的既定功能和任务。它是向未来的用户表明系统能够像预定要求那样工作，应检查软件能否按合同要求进行工作，即是否满足软件需求说明书中的确认标准。

验收测试是以用户为主的测试。软件开发人员和QA(质量保证)人员也应参加。由用户参加设计测试用例，在用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试，在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

验收测试同样需要制订测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确，人机界面和其他方面(例如，可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。验收测试的结果有两种可能，一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法，决定必须作很大修改还是在维护后期或下一个版本改进。

验收测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。

实施验收测试既可以是非正式的测试，也可以是有计划、有系统的测试。在软件交付使用之后，用户将如何实际使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误解、异常的数据组合以及产生对某些用户来说是清晰的但对另一些用户来说却难以理解的输出等。由于一个软件产品可能拥有众多用户，不可能由每个用户都进行验收，而且初期验收测试中大量的错误可能导致开发延期，甚至失去用户，因此多采用一种称为α、β测试的过程，以发现可能只有最终用户才能发现的错误。

α测试是软件开发公司组织内部人员模拟各类用户对即将面世的软件产品(称为α版本)进行测试。这是在受控制的环境下进行的测试。它的关键在于要尽可能逼真地模拟实际运行环境和用户对软件产品的操作，并尽最大努力涵盖所有可能的用户操作方式。α测试人员是除产品开发人员之外首先见到产品的人，他们提出的功能和修改意见是特别有价值的。α测试可以从软件产品编码结束之时开始，或在模块(子系统)测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始，有关的手册(草稿)等应事先准备好。经过α测试调整的软件产品称为β版本。

β测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。与α测试不同的是，开发者通常不在测试现场。因而，β测试是在开发者无法控制的环境下进行的软件现场应用。在β测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告，开发者在综合用户的报告之后，做出修改，最后将软件产品交付给全体用

户使用。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。由于 β 测试的主要目标是测试可支持性，所以 β 测试应尽可能由主持产品发行的人员来管理。

2.9 面向对象软件测试

传统软件开发采用面向过程、面向功能的方法，将程序系统模块化，也产生相应的单元测试、集成测试等方法。面向对象软件测试的整体目标和传统软件测试是一致的，即以最小的工作量发现尽可能多的错误。其动态测试过程也与传统软件一样，分为制定测试计划、产生测试用例、执行测试和评价几个阶段。但面向对象的程序结构不再是传统的功能模块结构，类是构成面向对象程序的基本成分。在类定义中封装了数据(用于表示对象的状态)及作用在数据上的操作，数据和操作统称为特征。对象是类的实例，类和类之间按继承关系组成一个有向无圈图结构。父类中定义了共享的公共特征，子类除继承了父类中定义的所有特征外，还可以引入新的特征，也允许对继承的方法进行重定义。面向对象语言提供的动态绑定机制将对象与方法动态地联系起来，继承和动态绑定的结合使程序有较大的灵活性，当用户需求变动时，设计良好的面向对象程序变动相对较小。面向对象技术具有的信息隐蔽、封装、继承、多态和动态绑定等特性提高了软件开发的质量，但同时也给软件测试提出了新的问题，增加了面向对象软件测试的难度。

在面向对象的程序设计中，由于相同的语义结构，如类、属性、操作和消息，出现在分析、设计和代码阶段，因此，需要扩大测试的范围，重视面向对象分析和设计模式的复审。在分析阶段发现类属性定义中的问题，将可能减少和防止延伸到软件设计和软件编码阶段的错误，反之，若在分析阶段及设计阶段仍未检测到问题，则问题可能会传送到程序的编码过程当中，并造成耗费大量的开发资源。同时，测试的结果会造成对系统进行相关的修改，而修改有可能带来新的、更多的潜在问题。面向对象的分析和面向对象的设计提供了关于系统的结构和行为的实质性信息，因此，在产生代码前必须进行严格的复审。

软件测试层次是基于测试复杂性分解的思想，是软件测试的一种基本模式。传统层次测试基于功能模块的层次结构，而在面向对象软件测试中，继承和组装关系刻画了类之间的内在层次，它们既是构造系统结构的基础，也是构造测试结构的基础。面向对象程序的执行实际上是执行一个由消息连接起来的方法序列，而这个方法序列通常是由外部事件驱动的。面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。对面向对象程序的测试应当分为多少级别尚未达成共识。由于面向对象软件从宏观上来看是各个类之间的相互作用，类是面向对象方法中最重要的概念，是构成面向对象程序的基本成分，也是进行面向对象程序测试的关键，因此大部分文献都将类作为最小的可测试单元，得到一种较为普遍的面向对象软件测试层次划分方法，将面向对象程序测试分为三级：类级、类簇级和系统级。根据测试层次结构，面向对象软件测试总体上也呈现从单元级、集成级到系统级的分层测试结构，测试集成的过程是基于可靠部件组装系统的过程。

1. 面向对象软件的单元测试

传统的单元测试的对象是软件设计的最小单位——模块。单元测试应对模块内所有重要的控制路径设计测试用例,以便发现模块内部的错误。单元测试多采用白盒测试技术,系统内多个模块可以并行地进行测试。

当考虑面向对象软件时,单元的概念发生了变化。封装驱动了类和对象的定义,这意味着每个类和类的实例(对象)包装了属性(数据)和操纵这些数据的操作。一个类可以包含一组不同的操作,而一个特定的操作也可能存在于一组不同的类中。因此,单元测试的意义发生了较大变化,不再孤立地测试单个操作,而是将操作作为类的一部分。此时最小的可测试单位是封装的类或对象,而不再是个体的模块。

面向对象的单元测试通常也称为类测试。传统单元测试主要关注模块的算法实现和模块的接口间数据的传递,而 OO 的类测试主要考察封装在一个类中的方法和类的状态行为。进行类测试时要把对象与其状态结合起来,进行对象状态行为的测试,因为工作过程中对象的状态可能被改变,产生新的状态。而对象状态的正确与否取决于该对象自创建以来接收到的消息序列,以及该对象对这些消息序列所作的响应。一个设计良好的类应能对正确的消息序列作出正确的反应,并具有抵御错误序列的能力。因而类测试应着重考察类的对象对消息序列的响应和对象状态的正确性。它与传统单元测试的区别如图 2.1 所示。

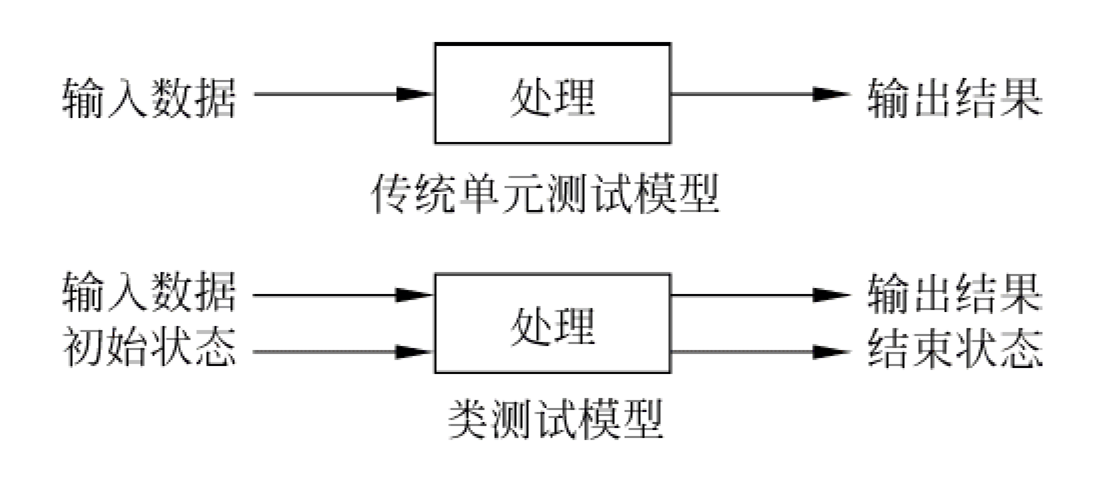


图 2.1 类测试与传统单元测试区别

2. 面向对象软件的集成测试

面向对象的集成测试即类簇测试。类簇是指一组相互有影响,联系比较紧密的类。它是一个相对独立的实体,在整体上是可执行和可测试的,并且实现了一个内聚的责任集合,但不提供被测试程序的全部功能,相当于一个子系统。类簇测试主要根据系统中相关类的层次关系,检查类之间的相互作用的正确性,即检查各相关类之间消息连接的合法性、子类的继承性与父类的一致性、动态绑定执行的正确性、类簇协同完成系统功能的正确性等。面向对象软件没有层次的控制结构,因此传统的自下而上或自上而下的集成测试策略并不适用于面向对象方法构造的软件,需要研究适合面向对象特征的新的集成测试策略。其测试有以下两种不同策略。

(1) 基于类间协作关系的横向测试。由系统的一个输入事件作为激励,对其触发的一组类进行测试,执行相应的操作/消息处理路径,最后终止于某一输出事件。应用回归测试对已测试过的类集再重新执行一次,以保证加入新类时不会产生意外的结果。

(2) 基于类间继承关系的纵向测试。首先通过测试不使用或很少使用其他类服务的类,即独立类(是系统中已经测试正确的某类)来开始构造系统。在独立类测试完成后,下一层继承独立类的类(称为依赖类)被测试,这个依赖类层次的测试序列一直循环执行到构造完整个系统。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注系统的结构和内部的相互作用。

3. 面向对象软件的系统测试

通过单元测试和集成测试，仅能保证软件开发的功能得以实现。但不能确认在实际运行时，它是否满足用户的需要。为此，对完成开发的软件必须经过规范的系统测试。系统测试是对所有程序和外部成员构成的整个系统进行整体测试，检验软件和其他系统成员配合工作是否正确。另外，还包括了确认测试内容，以验证软件系统的正确性和性能指标等是否满足需求规格说明书所制定的要求。它一般不考虑内部结构和中间结果，因此与传统的系统测试差别不大，可沿用传统的系统测试方法。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA (Object-Oriented Analysis)分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全“再现”问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。

练　习　题

1. 对软件测试的复杂性进行归纳分析。
2. 分别解释什么是静态测试、动态测试、黑盒测试、白盒测试、人工测试和自动化测试。
3. 如果没有软件规格说明或需求文档，可以进行动态黑盒测试吗？为什么？
4. 在单元测试中，所谓单元是如何划分的？
5. 简述单元测试的主要任务。
6. 如果开发时间紧迫，是否可以跳过单元测试而直接进行集成测试？为什么？
7. 什么是驱动模块和桩模块？为下面的函数构造一个驱动模块。

```
int divide(int a,int b)
{
  int c;
  if (b == 0){printf("除数不能为 0");return 0;}
  c = a/b;
  return c;
}
```

8. 什么是回归测试？什么时候进行回归测试？
9. 集成测试有哪些不同的集成方法？简述不同方法的特点。
10. 系统测试主要包括哪些内容？
11. 验收测试是由谁完成的？通常包含哪些过程？
12. 分析比较面向对象的软件测试与传统的软件测试的异同。

第3章 黑盒测试

3.1 黑盒测试法概述

黑盒测试又称为功能测试或数据驱动测试，着眼于程序外部结构，将被测试程序视为一个不能打开的黑盒子，完全不考虑程序内部逻辑结构和内部特性，主要针对软件界面、软件功能、外部数据库访问以及软件初始化等方面进行测试。因此黑盒测试的目的主要是在已知软件产品应具有功能的基础上，发现以下类型的错误。

(1) 检查程序功能是否按照需求规格说明书的规定正常使用，测试每个功能是否有遗漏，检测性能等特性是否满足要求。

(2) 检测人机交互是否错误，检测数据结构或外部数据库访问是否错误，程序是否能够适当地接收数据而产生正确的输出结果，并保持外部信息(如数据库或文件)的完整性。

(3) 检测程序初始化和终止方面的错误。

黑盒测试属于穷举输入测试方法，只有将所有可能的输入都作为测试情况来使用，才能检查出程序中所有的错误。但穷举测试是不现实的，因此需要选择合适的方法使设计出来的测试用例具有完整性、代表性，并能有效地发现软件缺陷。

黑盒测试常用的方法和技术主要包括边界值分析法、等价类划分法、决策表法、错误推测法、功能图法等。掌握和运用这些方法并不困难，但每种方法都有其所长，需要对被测软件的具体特点进行分析，选择合适的测试方法，才能有效解决软件测试中的问题。

3.2 边界值测试

3.2.1 边界值分析法

边界值分析法(boundary value analysis，BVA)是一种很实用的黑盒测试用例设计方法，它具有很强的发现程序错误的能力。无数的测试实践表明，大量的故障往往发生在输入定义域或输出值域的边界上，而不是在其内部，如做一个除法运算的例子，如果测试者忽略被除数为0的情况就会导致问题的遗漏。所以在设计测试用例时，一定要重视对边界值附近的处理。为检验边界附近的处理专门设计测试用例，通常都会取得很好的效果。

应用边界值分析的基本思想是：选取正好等于、刚刚大于和刚刚小于边界值的数据作为测试数据。边界值分析法是最有效的黑盒分析法，但在边界情况复杂时，要找出适当的边界测试用例还需要针对问题的输入域、输出域边界，耐心细致地逐个进行考察。

1. 边界值分析

边界值分析关注的是输入、输出空间的边界条件，以标识测试用例。实践证明，程序在处理大量中间数值时都正确，但在边界处可能出现错误。例如，循环条件漏写了等于，计数器少计了一次或多计了一次，数组下标忽略了 0 的处理等，这些都是平时编程容易疏忽而导致出错的地方。

刚开始，可能意识不到一组给定数据包含了多少边界，经过仔细分析总可以找到一些不明显的、有趣的或可能产生软件故障的边界。实际上，边界条件就是软件操作界限所在的边缘条件。

一些可能与边界有关的数据类型有：数值、速度、字符、位置、尺寸、数量等。同时，针对这些数据类型可以考虑它们的下述特征：第一个/最后一个，最小值/最大值，开始/完成，超过/在内，空/满，最短/最长，最慢/最快，最早/最迟，最高/最低，相邻/最远等。

以上是一些可能出现的边界条件。实际应用中，每一个软件测试问题都不完全相同，可能包含各式各样的边界条件，应视具体情况而定。

2. 内部边界值分析

上面边界值分析中所讨论的边界条件比较容易发现，它们在软件规格说明中或者有定义，或者可以在使用软件的过程中确定。而有些边界却是在软件内部，用户几乎看不到，但我们在进行软件测试时仍有必要对它们进行检查，这样的边界条件称为内部边界条件或次边界条件。

寻找内部边界条件比较困难，虽然不要求软件测试人员成为程序员或者具有阅读源代码的能力，但要求软件测试人员能大体了解软件的工作方式。例如，对文本输入或文本转换软件进行测试，在考虑数据区间包含哪些值时，最好参考一下 ASCII 表。如果测试的文本输入框只接受用户输入字符 A～Z 和 a～z，就应该在非法区间中，检查 ASCII 表中刚好位于 A 和 a 前面，Z 和 z 后面的值——“@”，“[”，“‘”和“{”。

3.2.2 边界值分析法测试用例

1. 边界值分析测试的基本思想

为便于理解，假设有两个变量 $x1$ 和 $x2$ 的函数 F，其中函数 F 实现为一个程序，$x1$、$x2$ 在下列范围内取值：

$$a \leqslant x1 \leqslant b, \quad c \leqslant x2 \leqslant d$$

区间$[a,b]$和$[c,d]$是 $x1$、$x2$ 的值域，程序 F 的输入定义域如图 3.1 所示，即带阴影矩形中的任何点都是程序 F 的有效输入。

采用边界值分析测试的基本原理是：故障往往出现在输入变量的边界值附近。例如，当一个循环条件为“≤”时，却错写成“<”，计时器少计数一次。

边界值分析测试的基本思想是在最小值(min)、略高于最小值(min+)、正常值(nom)、略低于最大值(max-)和最大值(max)处取输入变量值。同时，对于有多个输入变量的情况，通常是基于可靠性理论中称为“单故障”的假设，这种假设认为有两个或两个以上故障同时出现而导致软件失效的情况很少，也就是说，软件失效基本上是由单故障引起的。因此，边界值分析测试用例的获得，是通过使一个变量取极值，剩下所有变量取正常值。有两个输入变量的程序 F 的边界值分析测试用例如图 3.2 所示。

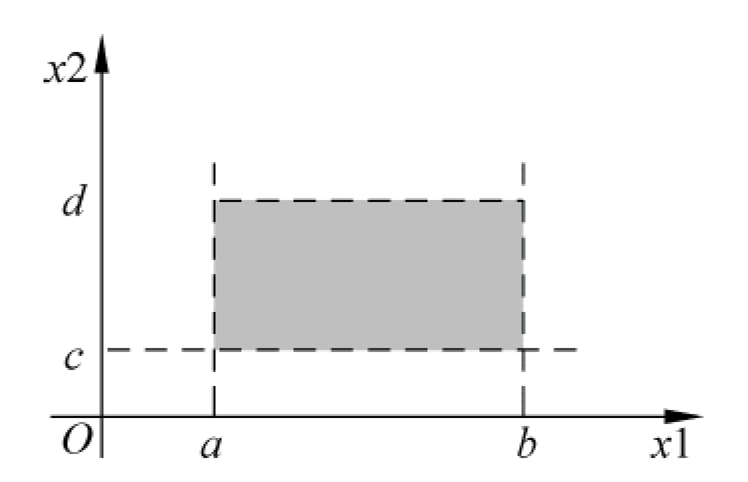


图 3.1　有两个变量 $x1$、$x2$ 的程序输入域

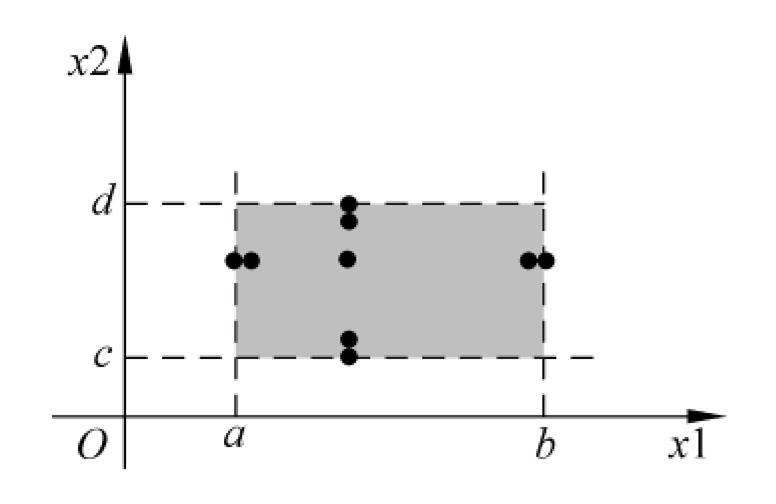


图 3.2　有两个输入变量的程序 F 的边界值分析测试用例

$\{<x1_{nom},x2_{min}>,<x1_{nom},x2_{min+}>,<x1_{nom},x2_{nom}>,<x1_{nom},x2_{max-}>,<x1_{nom},x2_{max}>,<x1_{min},x2_{nom}>,<x1_{min+},x2_{nom}>,<x1_{max-},x2_{nom}>,<x1_{max},x2_{nom}>\}$

对于一个含有 n 个变量的程序，保留其中一个变量，让其余的变量取正常值，被保留的变量依次取 min、min＋、nom、max－、max 值，对每个变量都重复进行。这样，对于一个有 n 个变量的程序，边界值分析测试程序会产生 $4n+1$ 个测试用例。如果没有显式的给出边界，如三角形问题，则必须创建一种人工边界，可以先设定下限值（边长应大于等于 1），并规定上限值，如 100，或取默认的最大可表示的整数值。

2. 健壮性测试

健壮性测试是边界分析测试的一种简单扩展，除了取 5 个边界值分析取值外，还需要考虑采用一个略超过最大值（max＋）以及略小于最小值（min－）的取值，检查超过极限值时系统的表现会是什么。健壮性测试最有意义的部分不是输入，而是预期的输出。它要观察例外情况如何处理，比如某个部分的负载能力超过其最大值可能出现的情形。健壮性测试用例如图 3.3 所示。

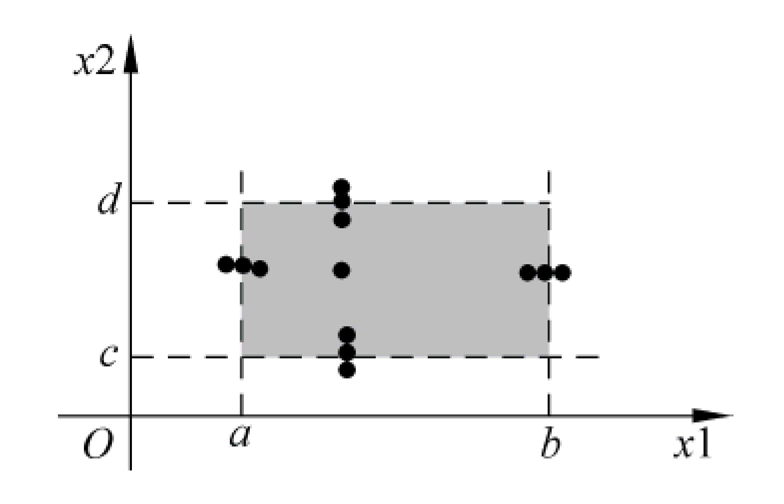


图 3.3　有两个输入变量的程序 F 的健壮性测试用例

3.2.3　边界值分析法测试实例

1. 三角形问题

问题描述如下。

三角形问题接受三个整数 a、b 和 c 作为输入，用做三角形的边。程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。

通过提供更多细节可以改进这个定义。于是这个问题变成以下的形式。

三角形问题接受三个整数 a、b 和 c 作为输入，用做三角形的边。整数 a、b 和 c 必须满足以下条件。

c1. $1\leqslant a\leqslant 100$　　c4. $a<b+c$

c2. $1\leqslant b\leqslant 100$　　c5. $b<a+c$

c3. $1\leqslant c\leqslant 100$　　c6. $c<a+b$

程序的输出是由这三条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。如果输入值没有满足 c1、c2 和 c3 这些条件中的任何一个，则程序会通过输出消息来进行通知，例如，“b 的取值不在允许取值的范围内。”如果 a、b 和 c 取值满足 c1、c2 和 c3，

则给出以下四种相互排斥输出中的一个。

(1) 如果三条边相等,则程序输出是等边三角形。

(2) 如果恰好有两条边相等,则程序输出是等腰三角形。

(3) 如果没有两条边相等,则程序输出是不等边三角形。

(4) 如果 c4、c5 和 c6 中有一个条件不满足,则程序输出是非三角形。

在三角形问题描述中,除了要求边长是整数外,没有给出其他的限制条件。边界下限为 1,上限为 100。表 3.1 给出了边界值分析测试用例。

表 3.1　三角形问题的边界值分析测试用例

测试用例	a	b	c	预 期 输 出
test1	60	60	1	等腰三角形
test2	60	60	2	等腰三角形
test3	60	60	60	等边三角形
test4	50	50	99	等腰三角形
test5	50	50	100	非三角形
test6	60	1	60	等腰三角形
test7	60	2	60	等腰三角形
test8	50	99	50	等腰三角形
test9	50	100	50	非三角形
test10	1	60	60	等腰三角形
test11	2	60	60	等腰三角形
test12	99	50	50	等腰三角形
test13	100	50	50	非三角形

2. NextDate 函数

问题描述: NextDate 是一个有三个变量(月份、日期和年)的函数。函数返回输入日期后面的那个日期。变量月份、日期和年都具有整数值,且满足以下条件。

c1. 1≤月份≤12

c2. 1≤日期≤31

c3. 1912≤年≤2050

在 NextDate 函数中,规定了变量 month、day、year 相应的取值范围,即 1≤month≤12, 1≤day≤31,1912≤year≤2050,表 3.2 给出了其健壮性测试用例。

表 3.2　NextDate 函数的边界分析测试用例

测试用例	month	day	year	预 期 输 出
test1	6	15	1911	year 超出[1912,2050]
test2	6	15	1912	1912. 6. 16
test3	6	15	1913	1913. 6. 16
test4	6	15	1975	1975. 6. 16
test5	6	15	2049	2049. 6. 16
test6	6	15	2050	2050. 6. 16
test7	6	15	2051	year 超出[1912,2050]

续表

测试用例	month	day	year	预 期 输 出
test8	6	－1	2001	day 超出[1,31]
test9	6	1	2001	2001.6.2
test10	6	2	2001	2001.6.3
test11	6	30	2001	2001.7.1
test12	6	31	2001	输入日期超界
test13	6	32	2001	day 超出[1,31]
test14	－1	15	2001	month 超出[1,12]
test15	1	15	2001	2001.1.16
test16	2	15	2001	2001.2.16
test17	11	15	2001	2001.11.16
test18	12	15	2001	2001.12.16
test19	13	15	2001	month 超出[1,12]

3.2.4 边界值分析局限性

如果被测程序是多个独立变量的函数，这些变量受物理量的限制，则很适合采用边界值分析。这里的关键是“独立”和“物理量”。

简单地看一下表 3.2 中 NextDate 函数的边界分析测试用例，就会发现其实这些测试用例是不充分的。例如，没强调 2 月和闰年。这里的真正问题是，月份、日期和年变量之间存在依赖关系，而边界值分析假设变量是完全独立的。不过即便如此，边界值分析也能够捕获月末和年末缺陷。边界值分析测试用例通过引用物理量的边界独立导出变量极值，不考虑函数的性质，也不考虑变量的语义含义。因此把边界值分析测试用例看作是初步的，这些测试用例的获得基本没有利用理解和想象。

物理量准则也很重要。如果变量引用某个物理量，例如温度、压力、空气速度、负载等，则物理边界极为重要。例如，菲尼克斯的航空港国际机场 1992 年 6 月 26 日被迫关闭，原因是当天的空气温度达到 122℉ 导致飞行员在起飞之前不能设置某一特定设备，因为该设备能够接受的最大空气温度是 120℉。

边界值分析对布尔变量和逻辑变量没有多大意义。例如布尔变量的极值是 true 和 false，但是其余 3 个值不明确。在后面章节可以看到，布尔变量可以采用基于决策表的测试。

3.3 等价类测试

使用等价类作为功能性能测试的基础有两个动机：希望进行完备的测试，同时又希望避免冗余。边界值测试不能实现这两种希望中的任意一个，研究那些测试用例表，很容易看出存在大量冗余，再进一步仔细研究，还会发现严重漏洞。等价类测试重复边界值测试的两个决定因素：健壮性和单/多缺陷假设。本节给出了 4 种形式的等价类测试，在弱/强等价类测试之分的基础之上针对是否进行无效数据的处理产生健壮与一般等价类测试之分。

3.3.1 等价类

等价类的重要特征是对它们构成集合的一个划分，其中，划分是指互不相交的一组子集，并且这些子集的并是整个集合。这对于测试有两点非常重要的意义：表示整个集合这个事实提供了一种形式的完备性，而互不相交可保证一种形式的无冗余性。由于子集是由等价关系决定的，因此子集的元素都有一些共同点。等价类测试的思想是通过每个等价类中的一个元素标识测试用例。如果广泛选择等价类，则可以大大降低测试用例之间的冗余。例如，在三角形问题中，首先要有一个等边三角形的测试用例，可能选择三元组(10,10,10)作为测试用例的输入。如果这样做了，则可以预期不会从诸如(3,3,3)和(100,100,100)这样的测试用例中得到多少新东西，这些测试用例会以与第一个测试用例一样的方式进行"相同处理"，因此，这些测试用例是冗余。当在考虑结构性测试时，将会看到"相同处理"映射到"遍历相同的执行路径"。

等价类测试的关键就是选择确定类的等价关系。通常是通过预测可能的实现，考虑在现实中必须提供的功能操作来做出这种选择。我们将用一系列例子说明这一点，但是首先必须区分弱和强等价类测试。

为了便于理解，将讨论与有两个变量 $x1$ 和 $x2$ 的函数 F 联系起来。如果 F 实现为一个程序，则输入变量 $x1$ 和 $x2$ 将拥有以下边界以及边界内的等价区间。

$$a \leqslant x1 \leqslant d \quad 区间为[a,b),[b,c),[c,d]$$

$$e \leqslant x2 \leqslant g \quad 区间为[e,f),[f,g]$$

其中，方括号和圆括号分别表示闭区间和开区间的端点。$x1$ 和 $x2$ 的无效值是：$x1<a$，$x1>d$ 以及 $x2<e$，$x2>g$。

1. 弱一般等价类测试

弱一般等价类测试的用例设计通过使用每个等价类(区间)的一个变量值实现(单缺陷假设的作用)。对于上面给出的例子，可得到如图 3.4 所示的弱一般等价类测试用例。

这三个测试用例使用每个等价类中的一个值。事实上，永远都有等量的弱等价类测试用例，因为测试用例数对应各个输入变量等价区间个数的最大值。

2. 强一般等价类测试

强一般等价类测试基于多缺陷假设，它需要等价类笛卡儿积的每个元素对应的测试用例(如图 3.5 所示)。笛卡儿积可保证两种意义上的"完备性"，一是覆盖所有的等价类，二是覆盖所有可能的输入组合。

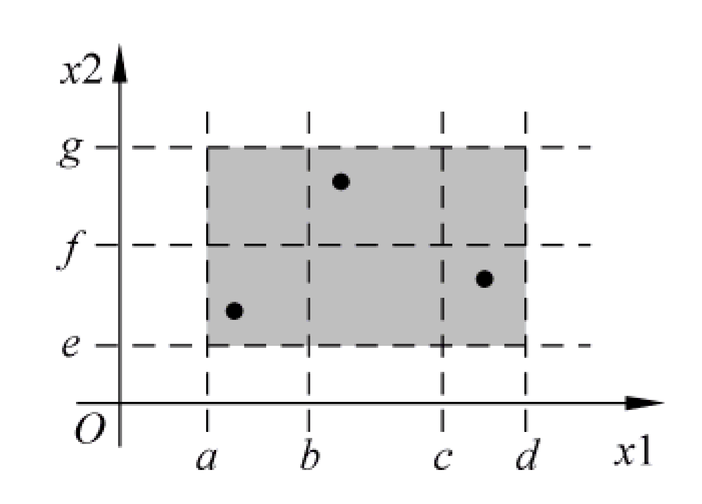


图 3.4　F 的弱一般等价类测试用例

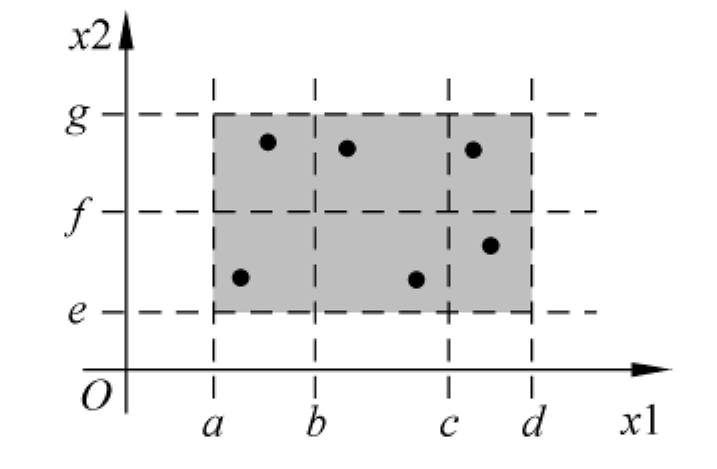


图 3.5　F 的强一般等价类测试用例

"好的"等价类测试的关键是等价关系的选择，要注意被"相同处理"的输入。在大多数情况下，等价类测试定义输入定义域的等价类。也可根据被测程序函数的输出值域定义等价关系，这对于三角形问题是最简单的方法。

3. 弱健壮等价类测试

这种测试的名称显然与直觉矛盾，怎么能够既弱又健壮呢？其实这是因为它是基于两个不同的角度而命名的，说它弱是因为它基于单缺陷假设，说它健壮是因为这种测试考虑了无效值。测试用例如图 3.6 所示。

(1) 对于有效输入，弱健壮等价类测试使用每个有效类的一个值，就像在弱一般等价类测试中所做的一样。请注意，这些测试用例中的所有输入都是有效的。

(2) 对于无效输入，弱健壮等价类测试的测试用例将拥有一个无效值，并保持其余的值都是有效的。此时，"单缺陷"会造成测试用例失败。

对于健壮等价类测试通常有两个问题。第一，规格说明常常并没有定义无效测试用例所预期的输出是什么，因此，测试人员需要花大量时间定义这些测试用例的输出。第二，强类型语言没有必要考虑无效输入。

4. 强健壮等价类测试

强健壮等价类测试，"强"是指该类测试用例的获得是基于多缺陷假设，"健壮"则和前面的定义一样，是指考虑了无效值。如图 3.7 所示，强健壮等价类测试从所有等价类笛卡儿积的每个元素中获得测试用例。

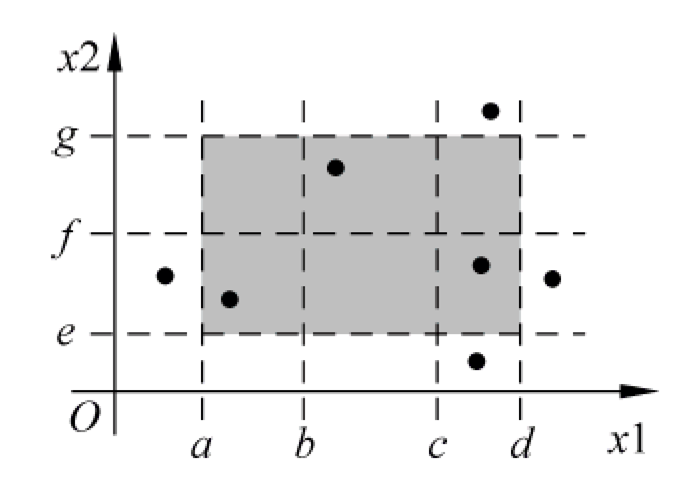


图 3.6　F 的弱健壮等价类测试用例

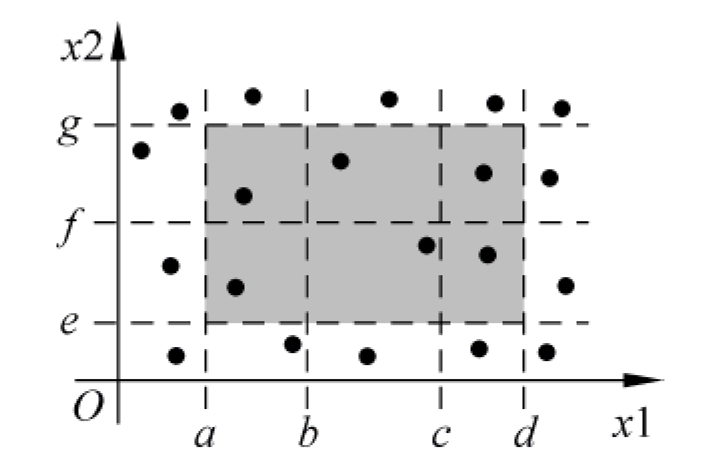


图 3.7　F 的强健壮等价类测试用例

3.3.2　等价类测试实例

1. 三角形问题

在描述问题时，曾经提到有 4 种可能出现的输出：非三角形、不等边三角形、等腰三角形和等边三角形。可以使用这些输出，标识如下所示的输出(值域)等价类。

$R1=\{<a,b,c>$：有三条边 a、b 和 c 的等边三角形$\}$

$R2=\{<a,b,c>$：有三条边 a、b 和 c 的等腰三角形$\}$

$R3=\{<a,b,c>$：有三条边 a、b 和 c 的不等边三角形$\}$

$R4=\{<a,b,c>$：有三条边 a、b 和 c 的非三角形$\}$

表 3.3 给出了 4 个弱一般等价类测试用例。

由于变量 a、b 和 c 没有有效区间划分，则强一般等价类测试用例与弱一般等价类测试用例相同。

考虑 a、b 和 c 的无效值，产生表 3.4 给出的弱健壮等价类测试用例。

表 3.3　三角形问题的弱一般等价类测试用例

测试用例	a	b	c	预期输出
WN1	5	5	5	等边三角形
WN2	2	2	3	等腰三角形
WN3	3	4	5	不等边三角形
WN4	4	1	2	非三角形

表 3.4　三角形问题的弱健壮等价类测试用例

测试用例	a	b	c	预期输出
WR1	−1	5	5	a 取值不在所允许的取值域内
WR2	5	−1	5	b 取值不在所允许的取值域内
WR3	5	5	−1	c 取值不在所允许的取值域内
WR4	201	5	5	a 取值不在所允许的取值域内
WR5	5	201	5	b 取值不在所允许的取值域内
WR6	5	5	201	c 取值不在所允许的取值域内

表 3.5 列出了额外强健壮等价类测试用例三维立方的一个“角”。

表 3.5　额外强健壮等价类测试用例三维立方的一个“角”

测试用例	a	b	c	预期输出
SR1	−1	5	5	a 取值不在所允许的取值域内
SR2	5	−1	5	b 取值不在所允许的取值域内
SR3	5	5	−1	c 取值不在所允许的取值域内
SR4	−1	−1	5	a、b 取值不在所允许的取值域内
SR5	5	−1	−1	b、c 取值不在所允许的取值域内
SR6	−1	5	−1	a、c 取值不在所允许的取值域内
SR7	−1	−1	−1	a、b、c 取值不在所允许的取值域内

请注意，预期输出如何完备地描述无效输入值。

等价类测试显然对于用来定义的等价关系很敏感。如果在输入定义域上定义等价类，则可以得到更丰富的测试用例集合。整数 a、b 和 c 可能取什么值呢？这些整数相等（有 3 种相等方式），或都不相等。

$$D1=\{<a,b,c>: a=b=c\}$$

$$D2=\{<a,b,c>: a=b, a\neq c\}$$

$$D3=\{<a,b,c>: a=c, a\neq b\}$$

$$D4=\{<a,b,c>: b=c, a\neq b\}$$

$$D5=\{<a,b,c>: a\neq b, a\neq c, b\neq c\}$$

作为一个单独的问题，可以通过三角形的性质来判断三条边是否构成一个三角形（例如，三元组<1,4,1>有一对相等的边，但是这些边不构成一个三角形）

$$D6=\{<a,b,c>: a\geqslant b+c\}$$

$$D7 = \{<a,b,c>: b \geqslant a+c\}$$
$$D8 = \{<a,b,c>: c \geqslant a+b\}$$

如果要彻底一些，可以将“小于或等于”分解为两种不同的情况，这样 $D6$ 就变成：

$$D6' = \{<a,b,c>: a = b+c\}$$
$$D6'' = \{<a,b,c>: a > b+c\}$$

同样对于 $D7$ 和 $D8$ 也有类似的情况。

2. NextDate 函数

NextDate 函数可以很好地说明选择内部等价关系的工艺。前面已经介绍过，NextDate 是一个三变量函数，即月份、日期和年，这些变量的有效值区间定义如下。

M1={月份：1≤月份≤12}

D1={日期：1≤月份≤31}

Y1={年：1812≤月份≤2012}

无效等价类如下。

M2={月份：月份<1}

M3={月份：月份>12}

D2={日期：日期<1}

D3={日期：日期>31}

Y2={年：年<1812}

Y3={年：年>2012}

由于每个独立变量的有效区间均为 1 个，因此只有弱一般等价类测试用例出现，并且与强一般等价类测试用例相同，如表 3.6 所示。

表 3.6　NextDate 函数的弱一般等价类测试用例

用例 ID	月份	日期	年	预 期 输 出
WN1,SN1	6	15	1912	1912 年 6 月 16 日

表 3.7 给出了其弱健壮测试用例的完整集合。

表 3.7　NextDate 函数的弱健壮等价类测试用例

用例 ID	月份	日期	年	预 期 输 出
WR1	6	15	1912	1912 年 6 月 16 日
WR2	−1	15	1912	月份不在有效值域 1～12 中
WR3	13	15	1912	月份不在有效值域 1～12 中
WR4	6	−1	1912	日期不在有效值域 1～31 中
WR5	6	32	1912	日期不在有效值域 1～31 中
WR6	6	15	1811	年不在有效值域 1812～2012 中
WR7	6	15	2013	年不在有效值域 1812～2012 中

与三角形问题一样，以下是额外强健壮性等价类测试用例三维立方的一个“角”，如表 3.8 所示。

表 3.8　NextData 函数的强健壮性等价测试用例三维立方的一个“角”

用例 ID	月份	日期	年	预 期 输 出
SR1	−1	15	1912	月份不在有效值域 1～12 中
SR2	6	−1	1912	日期不在有效值域 1～31 中
SR3	6	15	1811	年不在有效值域 1812～2012 中
SR4	−1	−1	1912	月份不在有效值域 1～12 中 日期不在有效值域 1～31 中
SR5	6	−1	1811	日期不在有效值域 1～31 中 年不在有效值域 1812～2012 中
SR6	−1	15	1811	月份不在有效值域 1～12 中 年不在有效值域 1812～2012 中
SR7	−1	−1	1811	月份不在有效值域 1～12 中 日期不在有效值域 1～31 中 年不在有效值域 1812～2012 中

划分等价关系的重点是等价类中的元素要被“同样处理”。上述方法所得测试用例集其实是不足的，因为它只注意到在单个变量处理的有效/无效层次上进行，而没有进一步分析具体处理的过程与特征。对该函数如果更仔细地选择等价关系，所得到的等价类和测试用例集将会更有用。

例如在 NextDate 函数中，注意到必须对输入日期做怎样的处理？如果它不是某个月的最后一天，则 NextDate 函数会直接对日期加 1；到了月末，下一个日期是 1，月份加 1；到了年末，日期和月份会复位到 1，年加 1。最后，闰年问题要确定有关的月份的最后一天。经过这些分析之后，可以假设有以下等价类。

M1＝{月份：每月有 30 天}

M2＝{月份：每月有 31 天}

M3＝{月份：此月是 2 月}

D1＝{日期：1≤日期≤28}

D2＝{日期：日期＝29}

D3＝{日期：日期＝30}

D4＝{日期：日期＝31}

Y1＝{年：年＝2000}

Y2＝{年：年是闰年}

Y3＝{年：年是平年}

通过选择有 30 天的月份和有 31 天的月份的独立类，可以简化月份最后一天问题。通过把 2 月分成独立的类，可以对闰年问题给予更多关注。还要特别关注日期的值：D1 中的日(差不多)总是加 1，D4 中的日只对 M2 中的月才有意义。最后，年有 3 个类，包括 2000 年这个特例、闰年和非闰年类。这并不是完美的等价类集合，但是通过这种等价类集合可以发现很多潜在错误。

这些类产生以下弱等价类测试用例，如表 3.9 所示。与前面一样，机械地从对应类的取值范围中选择输入。

表 3.9 弱等价类测试用例

用例 ID	月份	日期	年	预期输出
WN1	6	14	2000	2000 年 6 月 15 日
WN2	7	29	1996	1996 年 7 月 30 日
WN3	2	30	2002	不可能的输入日期
WN4	6	31	2000	不可能的输入日期

机械选择输入值不考虑领域知识,因此没有考虑两种不可能出现的日期。“自动”测试用例生成永远都会有这种问题,因为领域知识不是通过等价类选择获得的。

表 3.10 给出了经过改进的强一般等价类测试用例。

表 3.10 经过改进的强一般等价类测试用例

用例 ID	月份	日期	年	预期输出
SN1	6	14	2000	2000 年 6 月 15 日
SN2	6	14	1996	1996 年 6 月 15 日
SN3	6	14	2002	2002 年 6 月 15 日
SN4	6	29	2000	2000 年 6 月 30 日
SN5	6	29	1996	1996 年 6 月 30 日
SN6	6	29	2002	2002 年 6 月 30 日
SN7	6	30	2000	2000 年 7 月 1 日
SN8	6	30	1996	1996 年 7 月 1 日
SN9	6	30	2002	2002 年 7 月 1 日
SN10	6	31	2000	无效的输入日期
SN11	6	31	1996	无效的输入日期
SN12	6	31	2002	无效的输入日期
SN13	7	14	2000	2000 年 7 月 15 日
SN14	7	14	1996	1996 年 7 月 15 日
SN15	7	14	2002	2002 年 7 月 15 日
SN16	7	29	2000	2000 年 7 月 30 日
SN17	7	29	1996	1996 年 7 月 30 日
SN18	7	29	2002	2002 年 7 月 30 日
SN19	7	30	2000	2000 年 7 月 31 日
SN20	7	30	1996	1996 年 7 月 31 日
SN21	7	30	2002	2002 年 7 月 31 日
SN22	7	31	2000	2000 年 8 月 1 日
SN23	7	31	1996	1996 年 8 月 1 日
SN24	7	31	2002	2002 年 8 月 1 日
SN25	2	14	2000	2000 年 2 月 15 日
SN26	2	14	1996	1996 年 2 月 15 日
SN27	2	14	2002	2002 年 2 月 15 日
SN28	2	29	2000	无效的输入日期
SN29	2	29	1996	1996 年 3 月 1 日
SN30	2	29	2002	无效的输入日期
SN31	2	30	2000	无效的输入日期

续表

用例 ID	月份	日期	年	预 期 输 出
SN32	2	30	1996	无效的输入日期
SN33	2	30	2002	无效的输入日期
SN34	2	31	2000	无效的输入日期
SN35	2	31	1996	无效的输入日期
SN36	2	31	2002	无效的输入日期

从弱一般测试转向强一般测试会产生一些边界值测试中出现的冗余问题。从弱到强的转换，不管是一般类还是健壮类，都是以等价类的叉积表示。3 个月份类乘以 4 个日期类乘以 3 个年类，产生 36 个强一般等价类测试用例。对每个变量加上 2 个无效类，得到 150 个强健壮等价类测试用例。

通过更仔细地研究年类，还可以精简测试用例集合。通过合并 Y1 和 Y3，把结果称做平年，则 36 个测试用例就会降低到 24 个。这种变化不再特别关注 2000 年，并会增加判断闰年的难度。需要在难度和能够从当前用例中了解到的内容之间做平衡综合考虑。

3. 佣金问题

问题描述：前亚利桑那州境内的一位步枪销售商销售密苏里州制造商制造的步枪机(lock)、枪托(stock)和枪管(barrel)。枪机卖 45 美元，枪托卖 30 美元，枪管卖 25 美元。销售商每月至少要售出一支完整的步枪，且生产限额是大多数销售商在一个月内可销售 70 个枪机、80 个枪托和 90 个枪管。每访问一个镇子之后，销售商都给密苏里州步枪制造商发出电报，说明在那个镇子中售出的枪机、枪托和枪管的数量。到了月末，销售商要发出机枪'−1'表示一个月结束。这样步枪制造商就知道当月的销售情况，并计算销售商的佣金如下：销售额不到(含)1000 美元的部分为 10%，1000(不含)～1800(含)美元的部分为 15%，超过 1800 美元的部分为 20%。佣金程序生成月份销售报告，汇总售出的枪机、枪托和枪管总数，销售商的总销售额以及佣金。

佣金问题的输入定义域，由于枪机、枪托和枪管的限制而被“自然地”划分。这些等价类也正是通过传统等价类测试所标识的等价类。第一个类是有效输入，其他两个类是无效输入。在佣金问题中，仅考虑输入定义域等价类产生的测试用例集合并不令人满意。通过进一步分析发现对佣金函数的输出值域定义等价类可以有效改进测试用例集合。

输入变量对应的有效类是：

$L1$＝{枪机：1≤枪机≤70}

$L2$＝{枪机＝−1}

$S1$＝{枪托：1≤枪托≤80}

$B1$＝{枪管：1≤枪管≤90}

输入变量对应的无效类是：

$L2$＝{枪机：枪机＝0 或枪机＜−1}

$L3$＝{枪机：枪机＞70}

$S2$＝{枪托：枪托＜1}

$S3$＝{枪托：枪托＞80}

$B2$＝{枪管：枪管＜1}

$B3$={枪管：枪管>90}

其中变量枪机还用做指示不再有电报的标记。当枪机等于−1时，While循环就会终止，总枪机、总枪托和枪管的值就会被用来计算销售额，进而计算佣金，因此对于变量枪机增加了第2个有效类$L2$。

根据上述等价类的划分，可得如表3.11所示佣金问题的弱一般等价类测试用例，这个测试用例同样也等于强一般等价类测试用例。

表3.11　佣金问题的弱一般等价类测试用例

用例ID	枪机	枪托	枪管	预期输出
WN1	4	5	9	55.5

表3.12列出了7个弱健壮测试用例。

表3.12　佣金问题的弱健壮测试用例

用例ID	枪机	枪托	枪管	预期输出
WR1	4	5	9	55.5
WR2	0	40	50	枪机值不在有效值域1～70中
WR3	71	40	50	枪机值不在有效值域1～70中
WR4	30	0	50	枪托值不在有效值域1～80中
WR5	30	81	50	枪托值不在有效值域1～80中
WR6	30	40	0	枪管值不在有效值域1～90中
WR7	30	40	91	枪管值不在有效值域1～90中

最后，表3.13给出了额外强健壮等价类测试用例三维立方的一个“角”。

表3.13　额外强健壮等价类测试用例三维立方的一个“角”

用例ID	枪机	枪托	枪管	预期输出
SR1	0	40	45	枪机值不在有效值域1～70中
SR2	35	0	45	枪托值不在有效值域1～80中
SR3	35	40	0	枪管值不在有效值域1～90中
SR4	0	0	45	枪机值不在有效值域1～70中 枪托值不在有效值域1～80中
SR5	0	40	0	枪机值不在有效值域1～70中 枪管值不在有效值域1～90中
SR6	35	0	0	枪托值不在有效值域1～80中 枪管值不在有效值域1～90中
SR7	0	0	0	枪机值不在有效值域1～70中 枪托值不在有效值域1～80中 枪管值不在有效值域1～90中

注意，对于强测试用例，不管是强一般测试用例还是强健壮测试用例，都只有一个是合理输入。如果确实担心错误案例，那么这就是很好的测试用例集合。但是这样很难确信佣金的计算部分没有问题。在本例中，可以通过对输出值域定义等价类来进一步完善测试。前面提到过，销售额是所售出的枪机、枪托和枪管数量的函数：

销售额＝45×枪机＋30×枪托＋25×枪管

我们可以根据销售额值域定义3个等价类：

S1＝{＜枪机,枪托,枪管＞：销售额≤1000}

S2＝{＜枪机,枪托,枪管＞：1000＜销售额≤1800}

S3＝{＜枪机,枪托,枪管＞：销售额＞1800}

由此得到的输出值域等价类测试用例如表3.14所示。

表3.14　输出值域等价类测试用例

测试用例	枪机	枪托	枪管	销售额	佣金
OR1	5	5	5	500	50
OR2	15	15	15	1500	175
OR3	25	25	25	2500	360

这些测试用例让人感觉到正在接触问题的重要部分。与弱健壮测试用例结合在一起，就可得到佣金问题的相当不错的测试。另外，可能还希望增加一些边界检查，只是为了保证从1000美元到1800美元的转移是正确的。

3.3.3　指导方针

上面已介绍了3个例子，最后讨论关于等价类测试的一些观察和等价类测试指导方针。

1. 显然，等价类测试的弱形式(一般或健壮)不如对应的强形式的测试全面。
2. 如果实现语言的强类型(无效值会引起运行时错误)，则没有必要使用健壮形式的测试。
3. 如果错误条件非常重要，则进行健壮形式的测试是合适的。
4. 如果输入数据以离散值区间和集合定义，则等价类测试是合适的。当然也适用于如果变量值越界就会出现故障的系统。
5. 通过结合边界值测试，等价类测试可得到加强(可以“重用”定义等价类的工作成果)。
6. 如果程序函数很复杂，则等价类测试是被指示的。在这种情况下，函数的复杂性可以帮助标识有用的等价类，就像NextDate函数一样。
7. 强等价类测试假设变量是独立的，相应的测试用例相乘会引起冗余问题。而如果存在依赖关系，则常常会生成“错误”测试用例，就像NextDate函数一样(此时最好采用决策表技术解决)。
8. 在发现“合适”的等价关系之前，可能需要进行多次尝试，就像NextDate函数例子一样。如果不能肯定存在在“明显”或“自然”等价关系，最好对任何合理的实现进行再次预测。

3.4　基于决策表的测试

在所有的功能性测试方法中，基于决策表的测试方法是最严格的，因为决策表具有逻辑严格性。

自从20世纪60年代初以来，决策表一直被用来表示和分析复杂逻辑关系。决策表很适合描述不同条件集合下采取行动的若干组合的情况。表3.15给出了基本决策表术语。

表 3.15　决策表的各个部分

桩	规则 1	规则 2	规则 3、4	规则 5	规则 6	规则 7、8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

决策表有 4 个部分：粗竖线左侧是桩部分；右侧是条目部分。粗横线的上面是条件部分，下面是行动部分。可以引用条件桩、条件条目、行动桩和行动条目。条目部分中的一列是一条规则。规则只是在规则的条件部分中指示的条件环境下要采取什么行动。在表 3.15 给出的决策表中，如果 c1、c2 和 c3 都为真，则采取行动 a1 和 a2。如果 c1 和 c2 都为真而 c3 为假，则采取行动 a1 和 a3。在 c1 为真 c2 为假的条件下采取行动 a4，此时规则中的 c3 条目叫做“不关心”条目。不关心条目有两种主要解释：条件无关或条件不适用。

如果有二叉条件(真/假，是/否，0/1)，则决策表的条件部分是旋转了 90 度的(命题逻辑)真值表。这种结构能够保证考虑了所有可能的条件的组合。如果使用决策表标识测试用例，那么决策表的这种完备性质能够保证一种完备的测试。所有条件都是二叉条件的决策表叫做有限条目决策表。如果条件可以有多个值，则对应的决策表叫做扩展条目决策表。

决策表被设计为说明性的，给出的条件没有特别的顺序，而且所选择的行动发生时也没有任何特定顺序。

1. 表示方法

为了使用决策表标识测试用例，可把条件解释为输入，把行动解释为输出。有时条件最终引用输入的等价类，行动引用被测软件的主要功能处理部分。这时规则就解释为测试用例。由于决策表可以机械地强制为完备的，因此可以有测试用例的完整集合。

产生决策表的方法可以有多种。

在表 3.16 所示的决策表中，给出了不关心条目和不可能规则使用的例子。正如第一条规则所指示，如果整数 a、b 和 c 不构成三角形，则不关心可能的相等关系。在规则 3、4 和 6 中，如果两对整数相等，则根据传递性，第三对整数也一定相等，因此这些规则不可能满足。

表 3.16　三角形问题决策表

c1：a、b、c 构成三角形?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2：$a=b$?	—	Y	Y	Y	Y	N	N	N	N
c3：$a=c$?	—	Y	Y	N	N	Y	Y	N	N
c4：$b=c$?	—	Y	N	Y	N	Y	N	Y	N
a1：非三角形	X								
a2：不等边三角形									X
a3：等腰三角形					X		X	X	
a4：等边三角形		X							
a5：不可能			X	X		X			

表 3.17 所示的决策表给出了有关表示方法的另一种考虑：条件的选择可以大大地扩展决策表的规模。这里将老条件（c1：a、b、c 构成三角形？）扩展为三角形特性的 3 个不等式的详细表示。如果有一个不等式不成立，则 3 个整数就不能构成三角形。还可以进一步扩展，因为不等式不成立有两种方式，一条边等于另外两条边的和或严格大于另外两条边的和。

表 3.17　经过修改的三角形问题决策表

c1：$a<b+c$?	F	T	T	T	T	T	T	T	T	T	T
c2：$b<a+c$?	—	F	T	T	T	T	T	T	T	T	T
c3：$c<a+b$?	—	—	F	T	T	T	T	T	T	T	
c4：$a=b$?	—	—	—	T	T	T	T	F	F	F	F
c5：$a=c$?	—	—	—	T	T	F	F	T	T	F	F
c6：$b=c$?	—	—	—	T	F	T	F	T	F	T	F
a1：非三角形	X	X	X								
a2：不等边三角形											X
a3：等腰三角形							X		X	X	
a4：等边三角形				X							
a5：不可能					X	X		X			

如果条件引用了等价类，则决策表会有一种典型的外观。如表 3.18 所示的决策表来自 NextDate 问题，引用了可能的月份变量相互排斥的可能性。由于一个月份就是一个等价类，因此不可能有两个条目同时为真的规则。不关心条目（—）的实际含义是“必须失败”。有些决策表使用者用 F 表示这一点。

表 3.18　带有相互排斥条件的决策表

条　　件	规则 1	规则 2	规则 3
c1：月份在 M1 中？	T	—	—
c2：月份在 M2 中？	—	T	—
c3：月份在 M3 中？	—	—	T
a1			
a2			
a3			

不关心条目的使用，对完整决策表的识别方式有微妙的影响。对于有限的条目决策表，如果有 n 个条件，则必须有 2^n 条规则。如果不关心条目实际地表明条件是不相关的，则可以按以下方法统计规则数：没有不关心条目的规则统计为 1 条规则；规则中每出现一个不关心条目，该规则数乘一次 2。表 3.17 所示决策表的规则条目数统计如表 3.19 所示。请注意，规则总数是 64（正好是应该得到的规则条数）。

表 3.19　表 3.17 所示规则条数统计的决策表

c1：$a<b+c$?	F	T	T	T	T	T	T	T	T	T	T
c2：$b<a+c$?	—	F	T	T	T	T	T	T	T	T	T
c3：$c<a+b$?	—	—	F	T	T	T	T	T	T	T	
c4：$a=b$?	—	—	—	T	T	T	T	F	F	F	F
c5：$a=c$?	—	—	—	T	T	F	F	T	T	F	F

续表

c6：$b=c$?	—	—	—	T	F	T	F	T	F	T	F
规则条数统计	32	16	8	1	1	1	1	1	1	1	1
a1：非三角形	X	X	X								
a2：不等边三角形											X
a3：等腰三角形							X		X	X	
a4：等边三角形				X							
a5：不可能					X	X		X			

如果将这种简化算法应用于表 3.18 所示的决策表，会得到如表 3.20 所示的规则条数统计。

表 3.20　带有相互排斥条件的决策表规则条数统计

条　　件	规则 1	规则 2	规则 3
c1：月份在 M1 中？	T	—	—
c2：月份在 M2 中？	—	T	—
c3：月份在 M3 中？	—	—	T
规则条数统计	4	4	4
a1			

应该只有 8 条规则，所以显然有问题。为了找出问题所在，应扩展所有 3 条规则，用可能的 T 或 F 替代“—”，如表 3.21 所示。

表 3.21　表 3.20 的扩展版本

条　　件	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4
c1：月份在 M1 中？	T	T	T	T	T	T	F	F	T	T	F	F
c2：月份在 M2 中？	T	T	F	F	T	T	T	T	T	F	T	F
c3：月份在 M3 中？	T	F	T	F	T	F	T	F	T	T	T	T
规则条数统计	1	1	1	1	1	1	1	1	1	1	1	1
a1												

请注意，所有条目都是 T 的规则有 3 条：规则 1.1、2.1 和 3.1。条目是 T、T、F 的规则有两条：规则 1.2 和 2.2。类似地，规则 1.3 和 3.2、2.3 和 3.3 也是一样的。如果去掉这种重复，最后可得到 7 条规则，缺少的规则是所有条件都是假的规则。这种处理的结果如表 3.22 所示，表中还给出了不可能出现的规则。

表 3.22　包含不可能出现的规则的相互排斥条件

条　　件	1.1	1.2	1.3	1.4	2.3	2.4	3.4
c1：月份在 M1 中？	T	T	T	T	F	F	F
c2：月份在 M2 中？	T	T	F	F	T	T	F
c3：月份在 M3 中？	T	F	T	F	T	F	T
规则条数统计	1	1	1	1	1	1	1
a1：不可能	X	X	X		X		

这种识别完备决策表的能力，使解决冗余性和不一致性方面处于很有利的地位，表 3.23 给出的决策表是冗余的，因为有 3 个条件则应该是 $2^3=8$ 条规则，此处却有 9 条规则。（规则 9 和规则 1～4 中某一条相同，是冗余规则。）

表 3.23　一个冗余决策表

条件	1～4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X

注意规则 9 的行为条目与规则 1～4 的条目相同。只要冗余规则中的行为与决策表相同的部分相同，就不会有什么大问题。如果行为条目不同，如表 3.24 所示的情况，则会遇到比较大的问题。

表 3.24　一个不一致的决策表

条件	1～4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

表 3.24 所示的决策表被用来处理事务，其中 c1 是真，c2 和 c3 都是假，则规则 4 和规则 9 都适用。可以观察到以下两点。

（1）规则 4 和规则 9 是不一致的，因为它们的行为集合是不同的。

（2）决策表是非确定的。因为此时不能确定是应该应用规则 4 还是规则 9。因此测试人员在应用决策表技术时要小心使用不关心条目。

2. 决策表的应用

决策表最为突出的优点是，能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏，因此，利用决策表能够设计出完整的测试用例集合。运用决策表设计测试用例，可以将条件理解为输入，将动作理解为输出。

（1）三角形问题的测试用例

使用表 3.17 所示的决策表，可得到 11 个功能性测试用例；3 个不可能测试用例；3 个测试用例违反三角形性质；1 个测试用例可得到等边三角形；1 个测试用例可得到不等边三角形；3 个测试用例可得到等腰三角形（如表 3.25 所示）。如果扩展决策表以显示两种违反三角形性质的方式，可以再选三个测试用例（一条边正好等于另外两条边的和）。做到这一点需要做一定的判断，否则规则会呈指数级增长。在这种情况下，最终会再得到很多不关心条目和不可能的规则。

表 3.25　根据表 3.17 得到的测试用例

用例 ID	a	b	c	预期输出
DT1	4	1	2	非三角形
DT2	1	4	2	非三角形
DT3	1	2	4	非三角形
DT4	5	5	5	等边三角形
DT5	?	?	?	不可能
DT6	?	?	?	不可能
DT7	2	2	3	等腰三角形
DT8	?	?	?	不可能
DT9	2	3	2	等腰三角形
DT10	3	2	2	等腰三角形
DT11	3	4	5	不等边三角形

（2）NextDate 函数测试用例

NextDate 函数可以说明定义域中的依赖性问题，决策表可以突出这种依赖关系，因此使得它成为基于决策表测试的一个完美例子。前面介绍过 NextDate 函数的等价类划分。等价类划分的不足之处是机械地选取输入值，可能会产生“奇怪”的测试用例，如找 2003 年 4 月 31 日的下一天。问题产生的根源是等价类划分和边界值分析测试都假设了变量是独立的。若变量之间在输入定义域中存在某种逻辑依赖关系，则这些依赖关系在机械地选取输入值时就可能会丢失。决策表方法通过使用“不可能动作”的概念表示条件的不可能组合，能够强调这种依赖关系。

为了产生给定日期的下一个日期，NextDate 函数能够使用的操作只有 5 种：day 变量和 month 变量的加 1 和复位操作，year 变量的加 1 操作。

在以下等价类集合上建立决策表。

M1：{month：month 有 30 天}

M2：{month：month 有 31 天，12 月除外}

M3：{month：month 是 12 月}

M4：{month：month 是 2 月}

D1：{day：1≤day≤27}

D2：{day：day=28}

D3：{day：day=29}

D4：{day：day=30}

D5：{day：day=31}

Y1：{year：year 是闰年}

Y2：{year：year 不是闰年}

表 3.26 所示是决策表，共有 22 条规则。

规则 1～5 处理有 30 天的月份，其中不可能规则也列出，如规则 5 处理在有 30 天的月份中考虑 31 日；规则 6～10 和规则 11～15 处理有 31 天的月份，其中规则 6～10 处理 12 月之外的月份，规则 11～15 处理 12 月；最后的 7 条规则关注 2 月和闰年问题。

表 3.26　NextDate 函数的决策表

选项＼规则	规则 1	规则 2	规则 3	规则 4	规则 5	规则 6	规则 7	规则 8	规则 9	规则 10	规则 11
条件：											
c1：month 在	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2	M3
c2：day 在	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5	D1
c3：year 在	—	—	—	—	—	—	—	—	—	—	—
动作：											
a1：不可能					√						
a2：day 加 1	√	√	√			√	√	√	√		√
a3：day 复位				√						√	
a4：month 加 1				√						√	
a5：month 复位											
a6：year 加 1											
选项＼规则	**规则 12**	**规则 13**	**规则 14**	**规则 15**	**规则 16**	**规则 17**	**规则 18**	**规则 19**	**规则 20**	**规则 21**	**规则 22**
条件：											
c1：month 在	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2：day 在	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3：year 在	—	—	—	—	—	Y1	Y2	Y1	Y2	—	—
动作：											
a1：不可能									√	√	√
a2：day 加 1	√	√	√		√	√					
a3：day 复位				√			√	√			
a4：month 加 1							√	√			
a5：month 复位				√							
a6：year 加 1				√							

可进一步简化这 22 条规则。若决策表中有两条规则的动作项相同，则一定至少有一个条件能够把这两条规则用不关心条件合并。例如，规则 1、2、3 都涉及有 30 天的月份 day 类 D1、D2 和 D3，并且它们的动作项都是 day 加 1，因此可以将规则 1、2、3 合并。类似地，有 31 天的月份的 day 类 D1、D2、D3 和 D4 也可合并，2 月的 D4 和 D5 也可合并。简化后的决策表如表 3.27 所示。

表 3.27　简化后的 NextDate 函数决策表

选项＼规则	规则 1～3	规则 4	规则 5	规则 6～9	规则 10	规则 11～14	规则 15	规则 16	规则 17	规则 18	规则 19	规则 20	规则 21、22
条件：													
c1：month 在	M1	M1	M1	M2	M2	M3	M3	M4	M4	M4	M4	M4	M4
c2：day 在	D1～D3	D4	D5	D1～D4	D5	D1～D4	D5	D1	D2	D2	D3	D3	D4、D5
c3：year 在	—	—	—	—	—	—	—	—	Y1	Y2	Y1	Y2	—
动作：													
a1：不可能			√									√	√

续表

选项 \ 规则	规则 1～3	规则 4	规则 5	规则 6～9	规则 10	规则 11～14	规则 15	规则 16	规则 17	规则 18	规则 19	规则 20	规则 21、22
a2：day 加 1	√			√		√			√				
a3：day 复位		√			√		√	√		√	√		
a4：month 加 1		√			√			√		√	√		
a5：month 复位							√						
a6：year 加 1							√						

根据简化后的表 3.27 所示的决策表，可设计测试用例，如表 3.28 所示。

表 3.28　测试用例表

测试可用例	month	day	year	预 期 输 出
Test1～Test3	9	16	2001	17/9/2001
Test4	9	30	2004	1/10/2004
Test5	9	31	2001	不可能
Test6～Test9	1	16	2004	17/1/2004
Test10	1	31	2001	1/2/2001
Test11～Test14	12	16	2004	17/12/2004
Test15	12	31	2001	1/1/2002
Test16	2	16	2004	17/2/2001
Test17	2	28	2004	29/2/2004
Test18	2	28	2001	1/3/2001
Test19	2	29	2004	1/3/2004
Test20	2	29	2001	不可能
Test21、Test22	2	30	2004	不可能

3. 决策表测试适用范围

每种测试方法都有适用的范围。基于决策表的测试可能对于某些应用程序，如 NextDate 函数十分有效，但是对于另一些应用程序（如佣金问题）就不是很有效。基于决策表测试通常适用于要产生大量决策的情况，如三角形问题，或在输入变量之间存在重要的逻辑关系的情况，如 NextDate 函数。

一般来说，决策表测试法适用于具有以下特征的应用程序。

(1) if-then-else 逻辑突出；

(2) 输入变量之间存在逻辑关系；

(3) 涉及输入变量子集的计算；

(4) 输入与输出之间存在因果关系。

在建立决策表的过程中不容易一步到位，第一次标识的条件和行动往往可能不那么令人满意，与其他技术一样，这时采用迭代会有所帮助。把第一次得到的结果作为铺路石，逐渐改进，直到得到满意的决策表。

3.5　错误推测法

错误猜测大多基于经验，需要从边界值分析等其他技术获得帮助。这种技术猜测特定软件类型可能发生的错误类型，并且设计测试用例查出这些错误。对有经验的工程师来说，错误猜测有时是最有效的发现 bug 的测试设计方法。为了更好地利用现成的经验，可以列出一个错误类型的检查表，帮助猜测错误在程序中可能发生的位置，提高错误猜测的有效性。

练　习　题

1. 分析黑盒测试方法的特点。

2. 健壮等价类测试与标准等价类测试的主要区别是什么？

3. 试用等价分类法测试党政管理系统中党员出生年月的输入设计是否符合要求，假设出生年月格式为 yyyymmdd。

4. 找零钱最佳组合：假设商店货品价格(R)皆不大于 100 元(且为整数)，若顾客付款在 100 元内(P)，求找给顾客之最少货币个(张)数？货币面值 50 元(N50)，10 元(N10)，5 元(N5)，1 元(N1)共 4 种。试根据边界值法设计测试用例。

5. 试为三角形问题中的直角三角形开发一个决策表和相应的测试用例。注意，会有等腰直角三角形。

6. 现有一个学生标准化考试批阅试卷，产生成绩报告的程序。其规格说明如下：程序的输入文件由一些有 80 个字符的记录组成，所有记录分为 3 组，如图 3.8 所示。

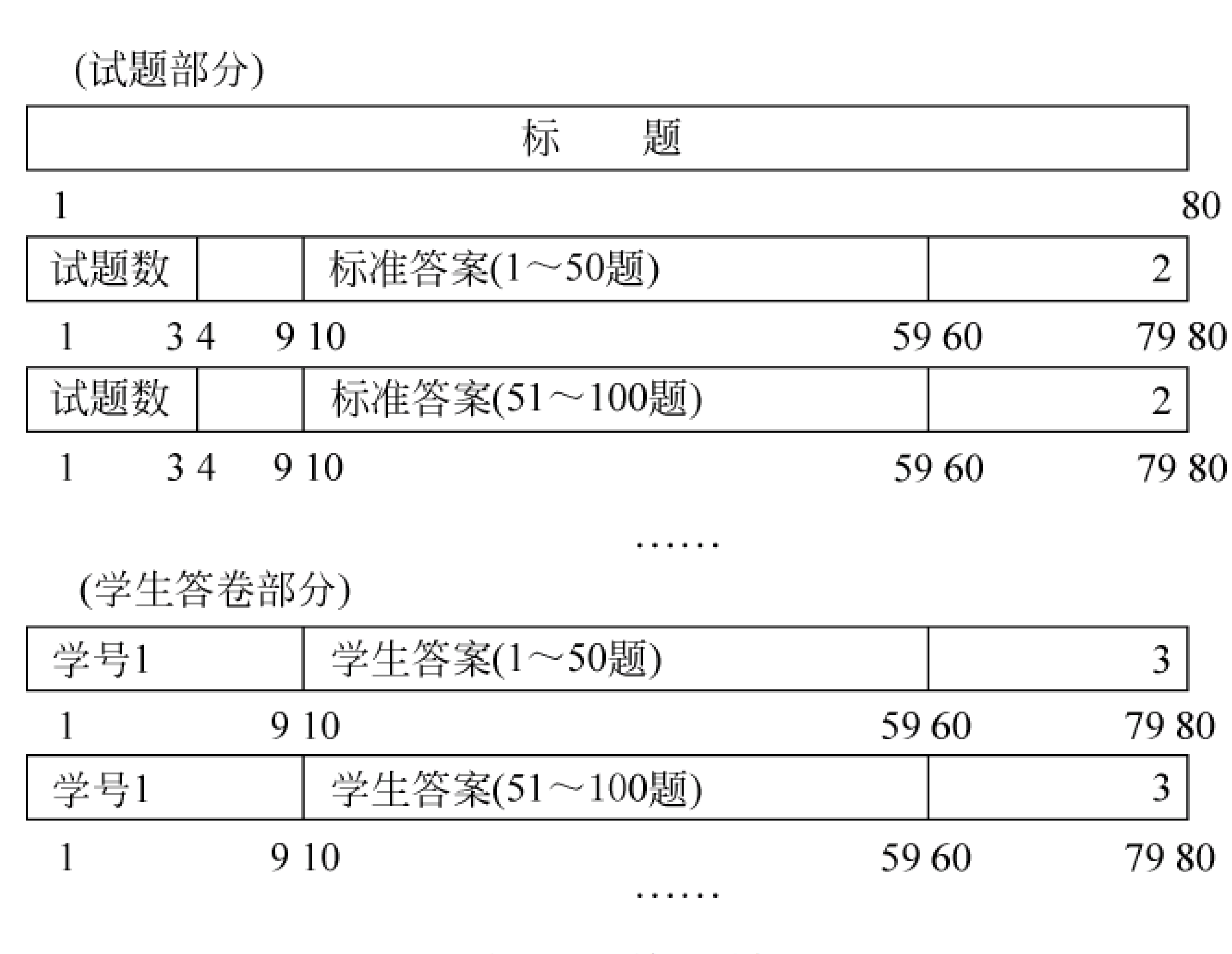


图 3.8　练习题 6

(1) 标题：该组只有一个记录，其内容是成绩报告的名字。

(2) 各题的标准答案：每个记录均在第 80 个字符处标以数字 2。该组的记录如下。

第一个记录：第 1～3 个字符为试题数(1～999)。第 10～59 个字符是 1～50 题的标准答案(每个合法字符表示一个答案)。

第二个记录：是第 51～100 题的标准答案。

……

(3) 学生的答案：每个记录均在第 80 个字符处标以数字 3。每个学生的答卷在若干个记录中给出。

学号：1～9 个字符。

1～50 题的答案：10～59。当大于 50 题时，在第二、三、……个记录中给出。

学生人数不超过 200，试题数不超过 999。

程序的输出有 4 个报告：

a) 按学号排列的成绩单，列出每个学生的成绩、名次。

b) 按学生成绩排序的成绩单。

c) 平均分数及标准偏差的报告。

d) 试题分析报告。按试题号排序，列出各题学生答对的百分比。

采用边界值分析方法，分析和设计测试用例。分别考虑输入条件和输出条件，以及边界条件。采用错误推测法补充设计一些测试用例。

第 4 章　白盒测试方法

白盒测试也称结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试，它的突出特点是基于被测程序的源代码，而不是软件的规格说明。在软件测试中，白盒测试一般是由程序员完成，当然也有专门做白盒测试的测试工程师。白盒测试人员必须对测试中的软件有深入的认识，包括其结构、各组成部分及之间的关联，以及其内部的运行原理、逻辑等。白盒测试人员实际上是程序员和测试员的结合体。

白盒测试的主要方法有程序结构分析、逻辑覆盖、基本路径测试等，它根据程序的控制结构设计导出测试用例，主要用于软件程序的验证。白盒测试法全面了解程序内部的逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法。在使用这种方法时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。

采用白盒测试方法必须遵循以下几条原则，才能达到测试的目的。

(1) 保证一个模块中的所有独立路径至少被测试一次。

(2) 所有逻辑值均需测试真和假两种情况。

(3) 检查程序的内部数据结构，保证其结构的有效性。

(4) 在上下边界及可操作范围内运行所有循环。

4.1　白盒测试基本概念

为了清晰描述白盒测试方法。需要首先对有关白盒测试的几个基本概念进行说明，即流图、环形复杂度和图矩阵。

1. 流图

在程序设计时，为了更加突出控制流的结构，可对程序流程图进行简化，简化后的图称为控制流图。

简化后产生的控制流图中所涉及的图形符号只有两种，即结点和控制流线。

(1) 结点用带有标号的圆圈表示，可以代表一个或多个语句、一个处理框程序和一个条件判断框(假设不包含复合条件)。

(2) 控制流线由带箭头的弧线或线表示，可称为边，它代表程序中的控制流。

常见语句的控制流图如图 4.1 所示。

包含条件的结点被称为判定结点(也叫谓词结点)，由判定结点发出的边必须终止于某一个相同结点，由边和结点所限定的范围被称为区域。

如果将一个典型的程序流程图转换为控制流图，转换结果如图 4.2 所示。

对于复合条件，则应将其分解为多个单个条件，并映射成控制流图，如图 4.3 所示，含复合条件的流程图(a)应首先分解为只含简单条件判断的流程图(c)，再转换得到控制流图(d)。

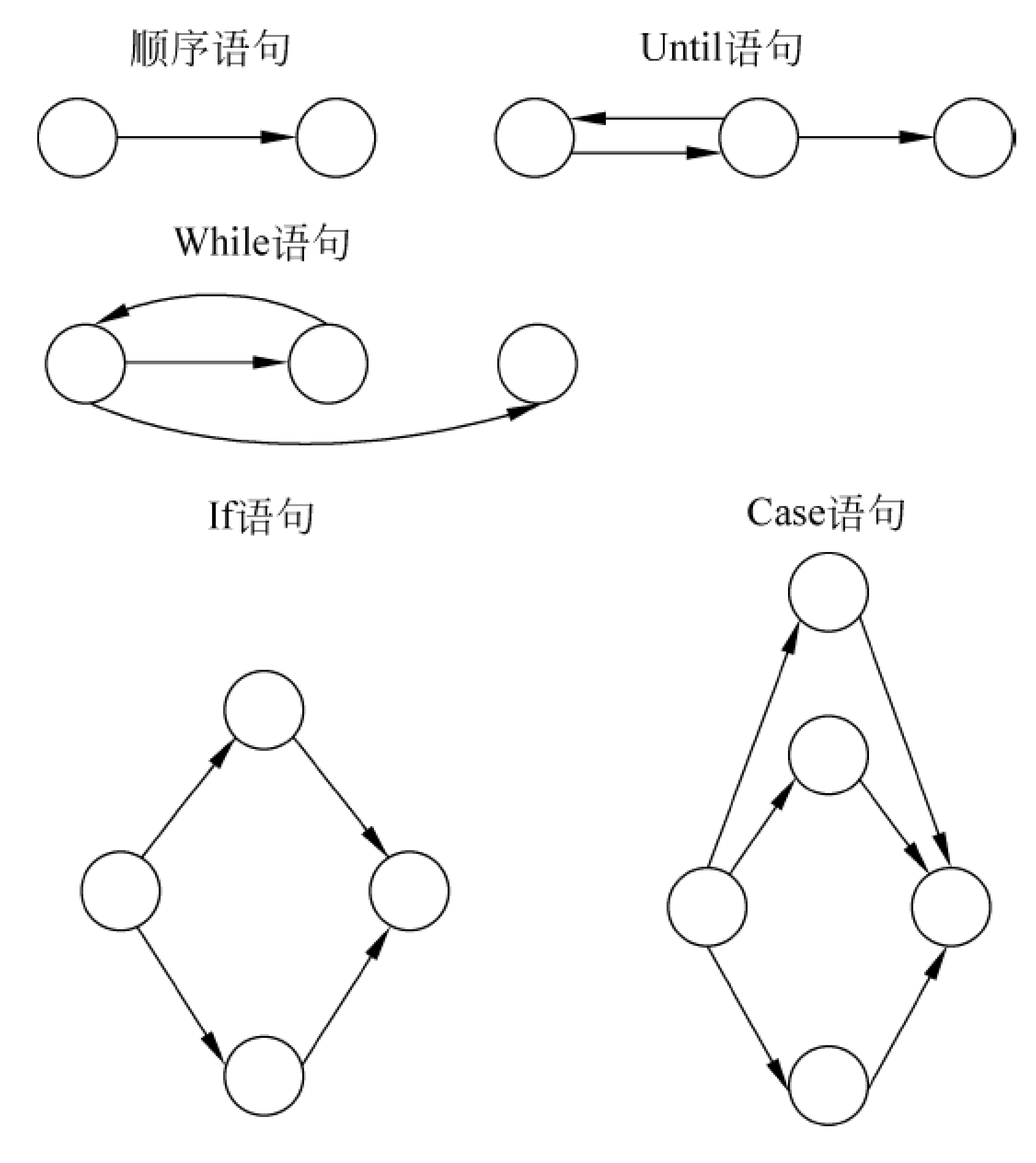


图 4.1　常见语句的控制流图

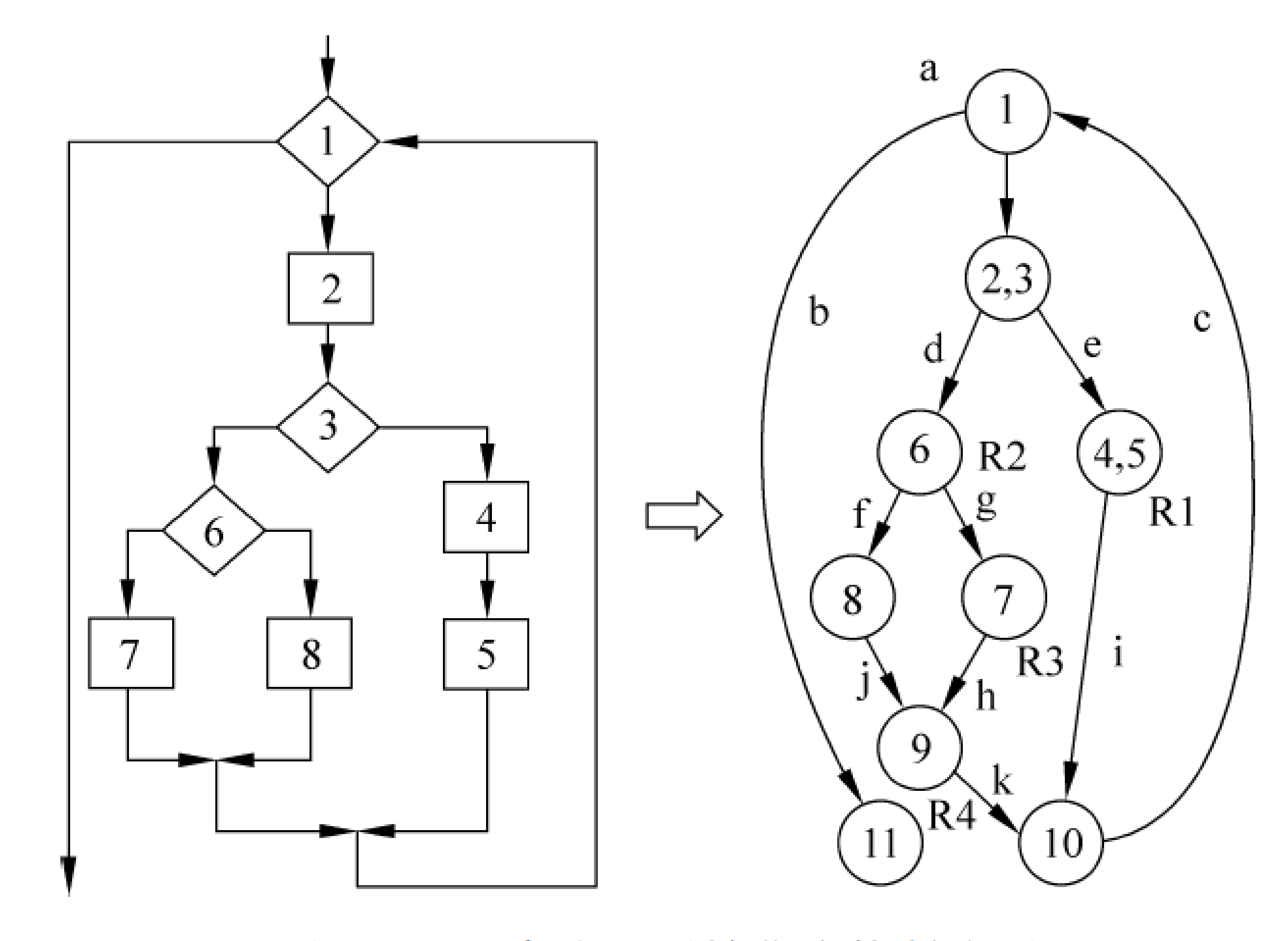


图 4.2　程序流程图转化为控制流图

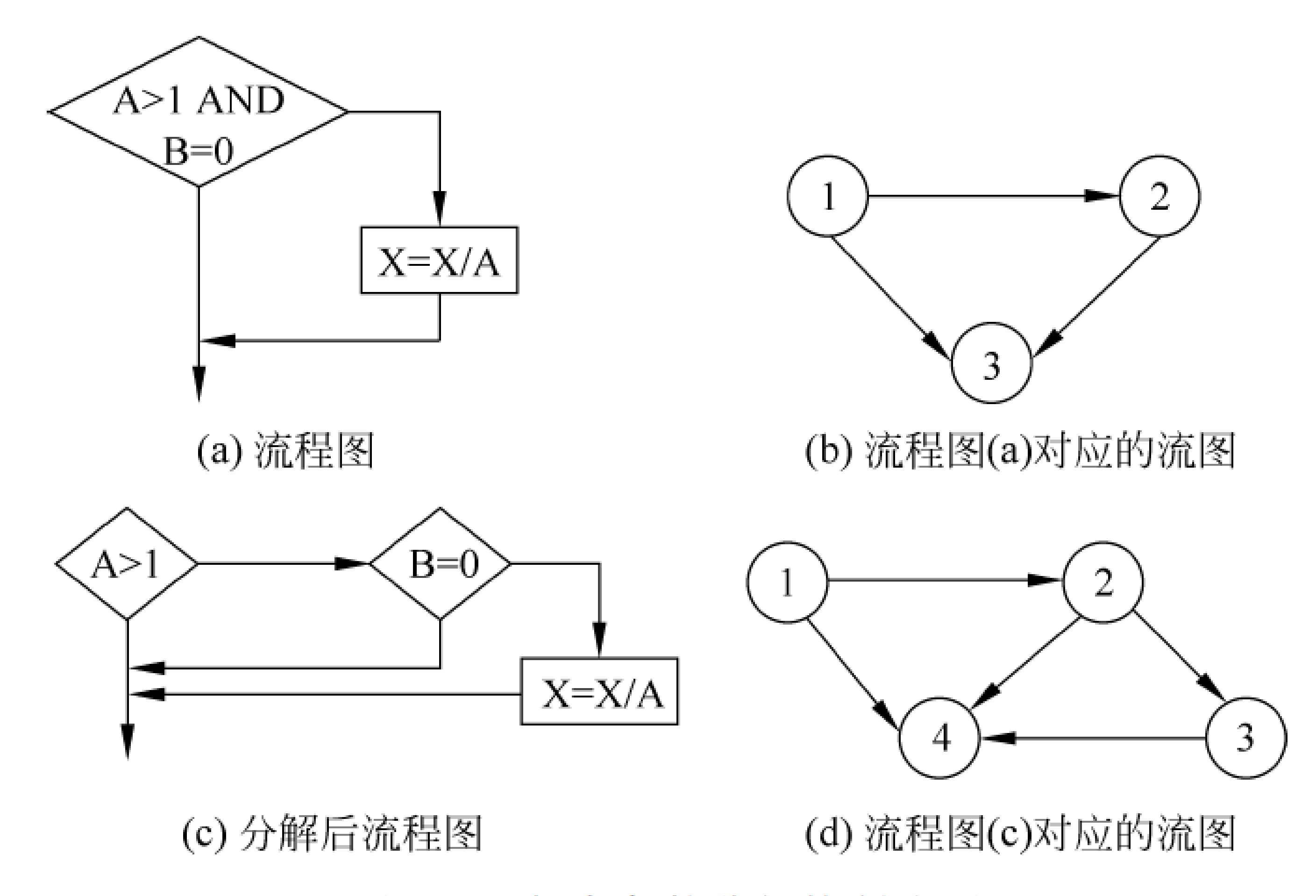


图 4.3　复合条件分解控制流图

2. 环形复杂度

环形复杂度也称为圈复杂度，概括地讲，它就是一种为程序逻辑复杂度提供定量尺度的软件度量。可以将该度量用于基本路径方法，它可以提供程序基本集的独立路径数量和确保所有语句至少执行一次的测试数量上界。其中，独立路径是指程序中至少引入一个新的处理语句集合或一个新条件的程序通路，它必须至少包含一条在本次定义路径之前不曾用过的边。路径可用流图中表示程序通路的结点序列表示，也可用弧线表示。

显而易见，程序中含有的路径数和程序的复杂性有着密切的关系，也就是说程序越复杂，它的路径数就越多。但程序复杂性如何度量呢？McCabe 给出了程序结构复杂性的计算公式。

程序控制流图是一个有向图，如果图中任何两个结点之间都至少存在一条路径，则这样的图称为强连通图。如果程序控制流图是一个强连通图，其复杂度 V(G)可按以下公式计算：

$$V(G) = e - n + 1$$

其中，e 为图 G 中的边数，n 为图 G 中的结点数，并且 McCabe 认为，强连通图的复杂度 V(G)就是图中线性独立环路的数量。

通过从汇结点到源结点添加一条边，便可创建控制流图的强连接有向图。如图 4.4 所示是一个经过了这种处理后的强连接有向图。其复杂度是：

$$V(G) = e - n + 1 = 11 - 7 + 1 = 5$$

图 4.4 中的强连接图的复杂度是 5，因此图 4.4 中有 5 个线性独立环路。如果现在删除从结点 G 到结点 A 所添加的边，则这 5 个环路就成为从结点 A 到结点 G 的线性独立路径。

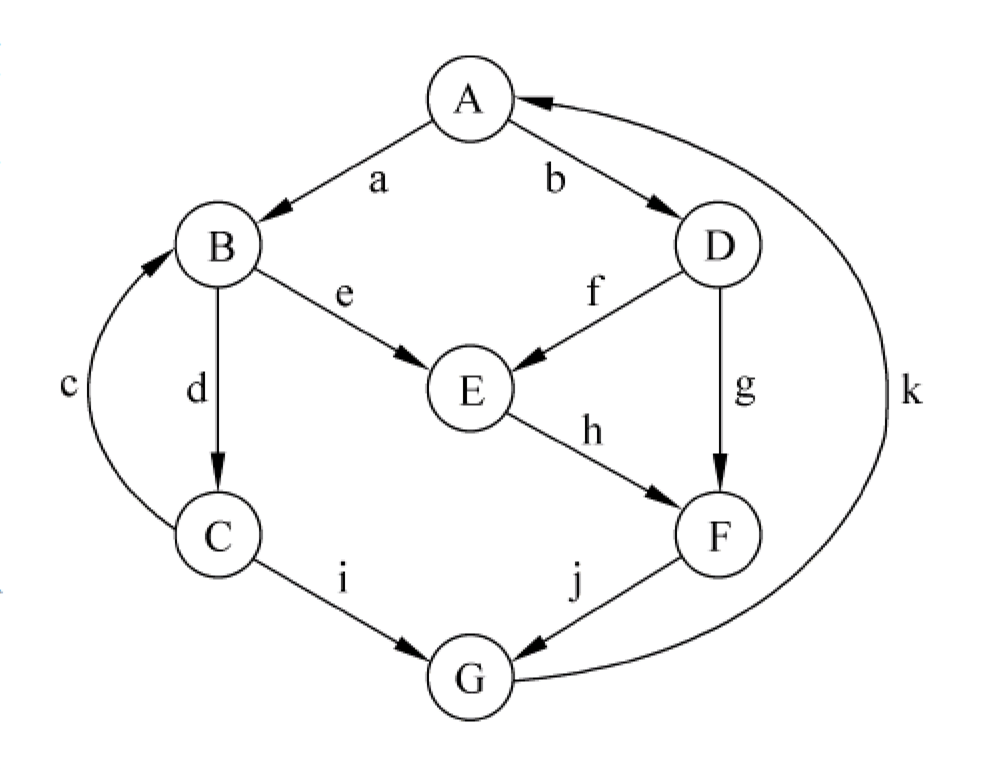


图 4.4 控制流图导出的强连通图

以下给出用结点序列表示的 5 条线性独立路径。

p1＝A，B，C，G

p2＝A，B，C，B，C，G

p3＝A，B，E，F，G

p4＝A，D，E，F，G

p5＝A，D，F，G

独立路径是指从程序入口到出口的多次执行中，每次至少有一个语句(包括运算、赋值、输入、输出或判断)是新的，未被重复的。如果用前面提到的控制流图来描述，独立路径就是在从入口进入控制流图后，至少要经历一条从未走过的弧。

因此，路径 p6＝A、B、C、B、E、F、G，p7＝A、B、C、B、C、B、C、G 不是独立路径。因为，p6 可以由路径 p1、p2 和 p3 组合而成，p7 可由路径 p1 和 p2 组合而成。

很明显，从测试角度来看，如果某一程序的每一条独立路径都测试过了，那么可以认为程序中的每个语句都已检验过了。但在实际测试中，要真正构造出程序的每条独立路径，并不是一件轻松的事。

测试可被设计为独立路径集，也称为基本路径集的执行过程。需要注意的是，基本路径集通常并不唯一。

3. 图矩阵

导出控制流图和决定基本测试路径的过程均需要机械化。为了开发辅助基本路径测试的软件工具,称为图矩阵的数据结构很有用,它可以用于实现自动地确定一个基本路径集。图矩阵即流图的邻接矩阵表示形式,其阶数等于流图的结点数。矩阵中的每列和每行都对应于标识的某一结点,矩阵元素对应于结点之间的边。如图 4.5 和图 4.6 所示,描述了一个简单的流图及其对应的矩阵。

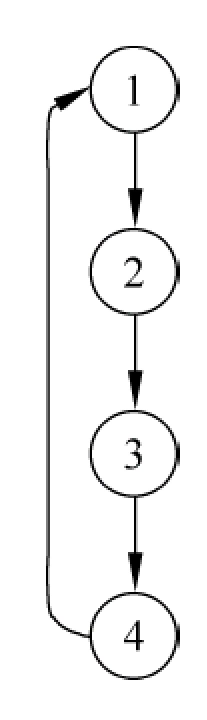


图 4.5　一个简单的流图

结点	1	2	3	4
1		a		
2			b	
3				c
4	d			

图 4.6　图 4.5 对应的邻接矩阵图

通常,流图中的结点用数字标识,边用字母标识。在如图 4.5 所示的例子当中,若矩阵记为 M,则 M(4,1)="d",表示边 d 连接结点 4 和结点 1。需要注意的是,边 d 是有方向的,它从结点 4 到结点 1。

4.2　逻辑覆盖

4.2.1　逻辑覆盖标准

有选择地执行程序中某些最有代表性的通路是对穷尽测试的唯一可行的替代办法。所谓逻辑覆盖是对一系列测试过程的总称,这组测试过程逐渐进行越来越完整的通路测试。测试数据执行(或叫覆盖)程序逻辑的程度可以划分成哪些不同的等级呢?从覆盖源程序语句的详尽程度分析,大致有以下一些不同的覆盖标准。

1. 语句覆盖

为了暴露程序中的错误,至少每个语句应该执行一次。语句覆盖的含义是,选择足够多的测试数据,使被测程序中每个语句至少执行一次。例如,图 4.7 所示的程序流程图描绘了一个被测试模块的处理算法。

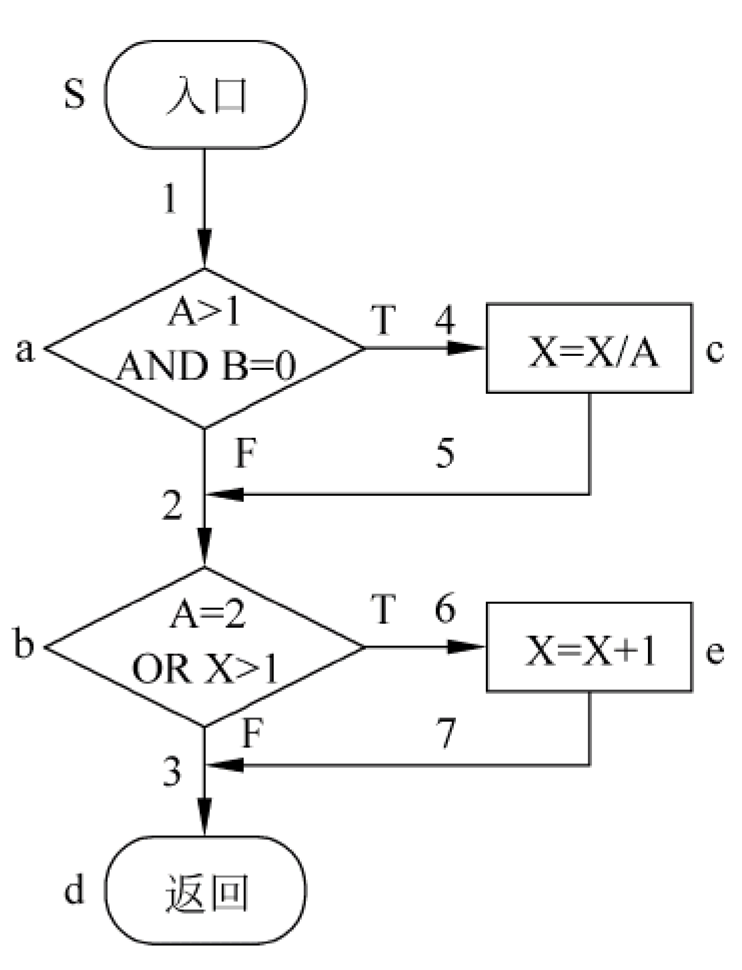


图 4.7　被测试模块的流程图

为了使每个语句都执行一次,程序的执行路径应该是 sacbed,为此只需要输入下面的测试数据(实际上 X 可以是任意实数):

$$A = 2,\quad B = 0,\quad X = 4$$

语句覆盖对程序的逻辑覆盖很少,在上面例子中两个判定条件都只测试了条件为真的情况,如果条件为假时处理有错误,显然不能发现。此外,语句覆盖只关心整个判

定表达式的值，而没有分别测试判定表达式中每个条件取值不同时的情况。在上面的例子中，为了执行 sacbed 路径，以测试每个语句，只需两个判定表达式(A＞1)AND(b＝0)和(A＝2)OR(X＞1)都取真值，因此使用上述一组测试数据就够了。但是，如果程序中把第一个判定表达式中的逻辑运算符“AND”错写成“OR”，或者把第二个判定表达式中的条件“X＞1”误写成“X＜1”，使用上面的测试数据并不能查出这些错误。

综上所述，可以看出语句覆盖是很弱的逻辑覆盖标准，为了更充分地测试程序，可以采用以下所述的逻辑覆盖标准。

2. 判定覆盖

判定覆盖又叫分支覆盖，它的含义是，不仅每个语句必须至少执行一次，而且每个判定表达式的每种可能的结果都应该至少执行一次，也就是每个判定的每个分支都至少执行一次。

对于上述例子来说，能够分别覆盖路径 sacbed 和 sabd 的两组测试数据，或者可以分别覆盖路径 sacbd 和 sabed 的两组测试数据，都满足判定覆盖标准。例如，用下面两组测试数据就可做到判定覆盖。

A＝3，B＝0，X＝3(覆盖 sacbd)

A＝2，B＝1，X＝1(覆盖 sabed)

判定条件覆盖比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高，例如，上面的测试数据只覆盖了程序全部路径的一半。

3. 条件覆盖

条件覆盖的含义是，不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。

如图 4.7 所示的例子总共有两个判定表达式，每个表达式中有两个条件，为了做到条件覆盖，应该选取测试数据使得在 a 点有下述各种结果出现：

$$A > 1, \quad A \leqslant 1, \quad B = 0, \quad B \neq 0$$

在 b 点有下述各种结果出现：

$$A = 2, \quad A \neq 2, \quad X > 1, \quad X \leqslant 1$$

只需要使用下面两组测试数据就可以达到上述覆盖标准。

Ⅰ. A＝2，B＝0，X＝4

(满足 A＞1，B＝0，A＝2 和 X＞1 的条件，执行路径 sacbed)

Ⅱ. A＝1，B＝1，X＝1

(满足 A≤1，B≠0，A≠2 和 X≤1 的条件，执行路径 sabd)

条件覆盖通常比判定覆盖强，因为它使判定表达式中每个条件都取到了两个不同的结果，判定覆盖却只关心整个判定表达式的值。例如，上面两组测试数据也同时满足判定覆盖标准。但是，也可能有相反的情况，虽然每个条件都取到了两个不同的结果，判定表达式却始终只取一个值。例如，如果使用下面两组测试数据，则只满足条件覆盖标准并不满足判定覆盖标准(第二个判定表达式的值总为真)。

Ⅰ. A＝2，B＝0，X＝1

(满足 A＞1，B＝0，A＝2 和 X≤1 的条件，执行路径 sacbed)

Ⅱ. $A=1,B=1,X=2$

(满足 $A\leqslant1,B\neq0,A\neq2$ 和 $X>1$ 的条件,执行路径 sabed)

4. 判定/条件覆盖

既然判定覆盖不一定包含条件覆盖,条件覆盖也不一定包含判定覆盖,自然会提出一种能同时满足这两种覆盖标准的逻辑覆盖,这就是判定/条件覆盖。它的含义是,选取足够多的测试数据,使得判定表达式中的每个条件都取到各种可能的值,而且每个判定表达式也都取到各种可能的结果。

对于如图 4.7 所示的例子而言,下述两组测试数据满足判定/条件覆盖标准。

Ⅰ. $A=2,B=0,X=4$

Ⅱ. $A=1,B=1,X=1$

但是,这两组测试数据也就是为了满足条件覆盖标准最初选取的两组数据,因此,有时判定/条件覆盖也并不比条件覆盖更强。

5. 条件组合覆盖

条件组合覆盖是更强的逻辑覆盖标准,它要求选取足够的测试数据,使得每个判定表达式中条件的各种可能组合都至少出现一次。

对于如图 4.7 所示的例子,共有以下 8 种可能的条件组合。

(1) $A>1,B=0$

(2) $A>1,B\neq0$

(3) $A\leqslant1,B=0$

(4) $A\leqslant1,B\neq0$

(5) $A=2,X>1$

(6) $A=2,X\leqslant1$

(7) $A\neq2,X>1$

(8) $A\neq2,X\leqslant1$

和其他逻辑覆盖标准中的测试数据一样,条件组合(5)~(8)中的 X 值是指在程序流程图第二个判定框(b 点)的 X 值。

下面的 4 个测试数据可以使上面列出的 8 种条件组合每种至少出现一次。

Ⅰ. $A=2,B=0,X=4$

(针对 1,5 两种组合,执行路径 sacbed)

Ⅱ. $A=2,B=1,X=1$

(针对 2,6 两种组合,执行路径 sabed)

Ⅲ. $A=1,B=0,X=2$

(针对 3,7 两种组合,执行路径 sabed)

Ⅳ. $A=1,B=1,X=1$

(针对 4,8 两种组合,执行路径 sabd)

显然,满足条件组合覆盖标准的测试数据,也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此,条件组合覆盖是前述几种覆盖标准中最强的。但是,满足条件组合覆盖标准的测试数据并不一定能使程序中的每一条路径都执行到,例如,上述 4 组测试数据都没有测试到路径 sacbd。

6. 路径覆盖

路径覆盖的定义是：选取足够多测试数据，使程序的每一条可能路径都至少执行一次。

对于如图 4.7 所示例子，请读者考虑满足路径覆盖的测试数据。

这里所用的程序段非常简短，只有 4 条路径。但在实际问题中，一个不太复杂的程序，其路径数都可能是一个庞大的数字，以致要在测试中覆盖所有的路径是不可能实现的。为解决这一难题，只得把覆盖的路径数压缩到一定限度内，例如，对程序中的循环体只执行一次。

即使对于路径数有限的程序做到了路径覆盖，也不能保证被测程序的正确性。因为通过分析测试数据，可以发现路径覆盖不能保证满足条件组合覆盖。而且在前面已经介绍过穷举路径测试法无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序，不可能查出程序因为遗漏路径而出现的错误，同时也发现不了一些与数据相关的错误。

由此看出，各种结构测试方法都不能保证程序的正确性。但是，测试的目的并不是要证明程序的正确性，而是要尽可能找出程序中隐藏的故障。事实上，并不存在一种十全十美的测试方法能够发现所有的软件故障。

4.2.2 最少测试用例数计算

为实现测试的完全逻辑覆盖，必须设计足够多的测试用例，并使用这些测试用例执行被测程序，实施测试。对某个具体程序来说，至少要设计多少测试用例。这里提供一种估算最少测试用例数的方法。

结构化程序是由 3 种基本控制结构组成。这 3 种基本控制结构就是：顺序型——构成串行操作；选择型——构成分支操作；重复型——构成循环操作。

为了把问题化简，避免出现测试用例极多的组合爆炸，把构成循环操作的重复型结构用选择结构代替。也就是说，并不指望测试循环体所有的重复执行，而是只对循环体检验一次。这样，任一循环便改造成进入循环体或不进入循环体的分支操作了。

如图 4.8 所示给出了类似于流程图的 N-S 图表示的基本控制结构（图中 A、B、C、D、S 均表示要执行的操作，P 是可取真假值的谓词，Y 表真值，N 表假值）。其中图 4.8(c)和图 4.8(d)两种重复型结构代表了两种循环。在简化如上循环的假设以后，对于一般的程序控制流，只考虑选择型结构。事实上它已能体现顺序型和重复型结构了。

如图 4.9 所示表达了两个顺序执行的分支结构。两个分支谓词 P1 和 P2 取不同值时，将分别执行 a 或 b 及 c 或 d 操作。显然，要测试这个小程序，需要至少提供 4 个测试用例才能做到逻辑覆盖，使得 ac、ad、bc 及 bd 操作均得到检验。其实，这里的 4 是由图 4.9 中第 1 个分支谓词引出的两个操作及第 2 个分支谓词引出的两个操作组合起来而得到的。

对于一般的、更为复杂的问题，估算最少测试用例数的原则也是同样的。现以图 4.10 表示的程序为例。该程序中共有 9 个分支谓词，尽管这些分支结构交错起来似乎十分复杂，很难一眼看出应至少需要多少个测试用例，但如果仍用上面的方法，也很容易解决。注意该图可分上下两层：分支谓词 1 的操作域是上层，分支谓词 8 的操作域是下层。这两层正像前面简单例子中的 P1 和 P2 的关系一样。只要分别得到两层的测试用例个数，再将其相乘即得总的测试用例数。这里需要首先考虑较为复杂的上层结构。谓词 1 不满足时要作的操

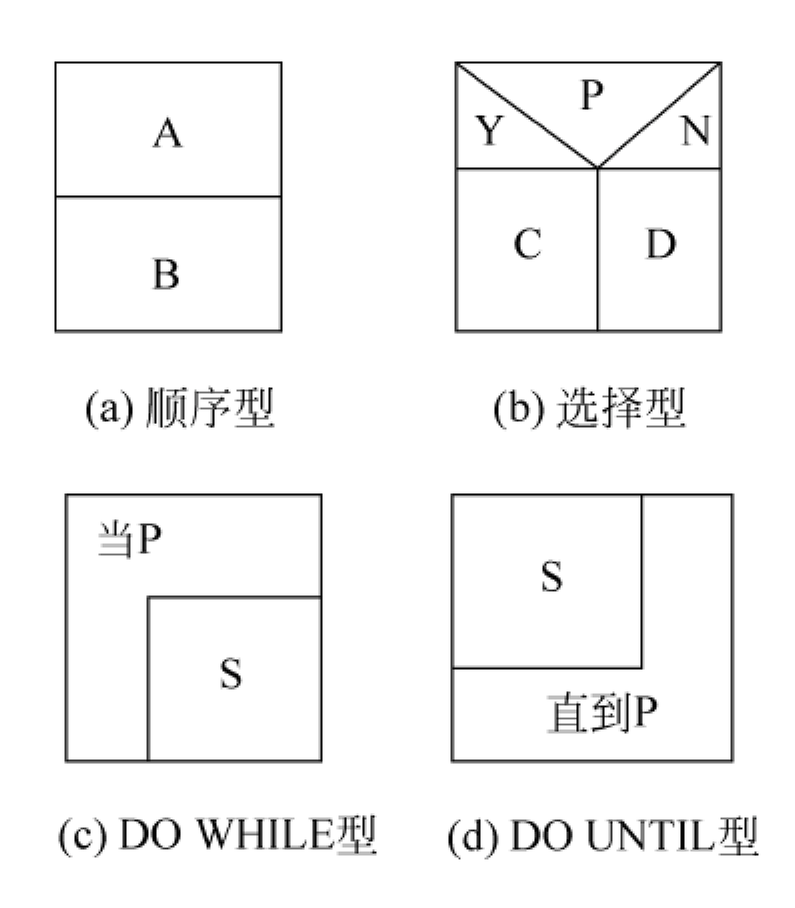


图 4.8　N-S 图表示的基本控制结构

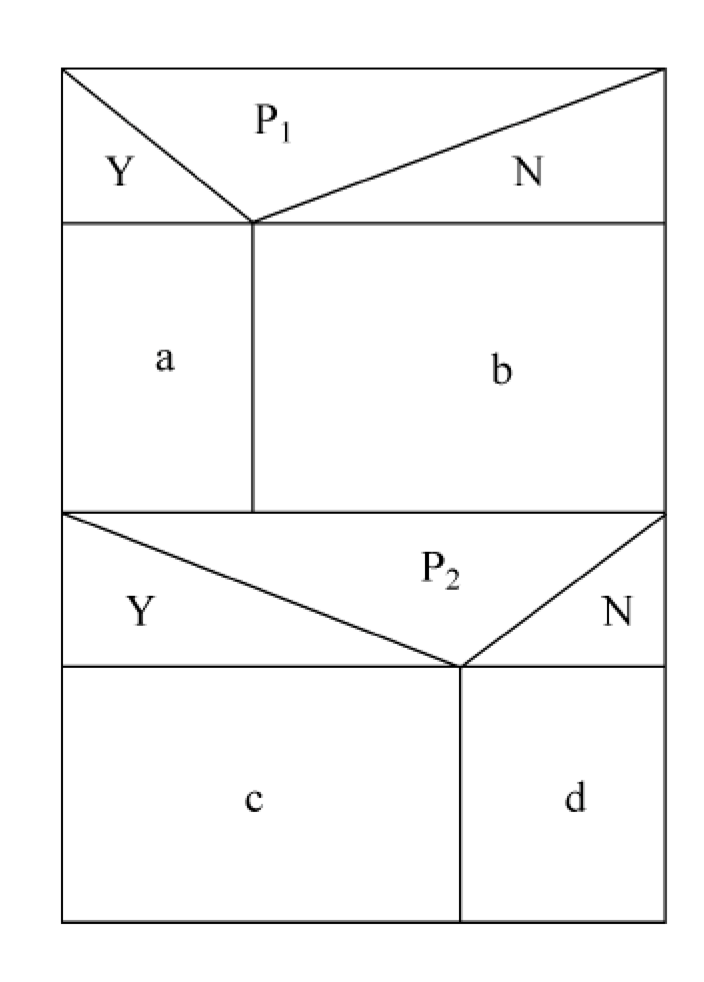


图 4.9　两个串行的分支结构的 N-S 图

作又可进一步分解为两层，这就是图 4.11 中的子图(a)和(b)。它们所需测试用例个数分别为 1+1+1+1+1=5 及 1+1+1=3，因而两层组合，得到 5×3=15。于是整个程序结构上层所需测试用例数为 1+15=16，而下层显然为 3。故最后得到整个程序所需测试用例数至少为 16×3=48。

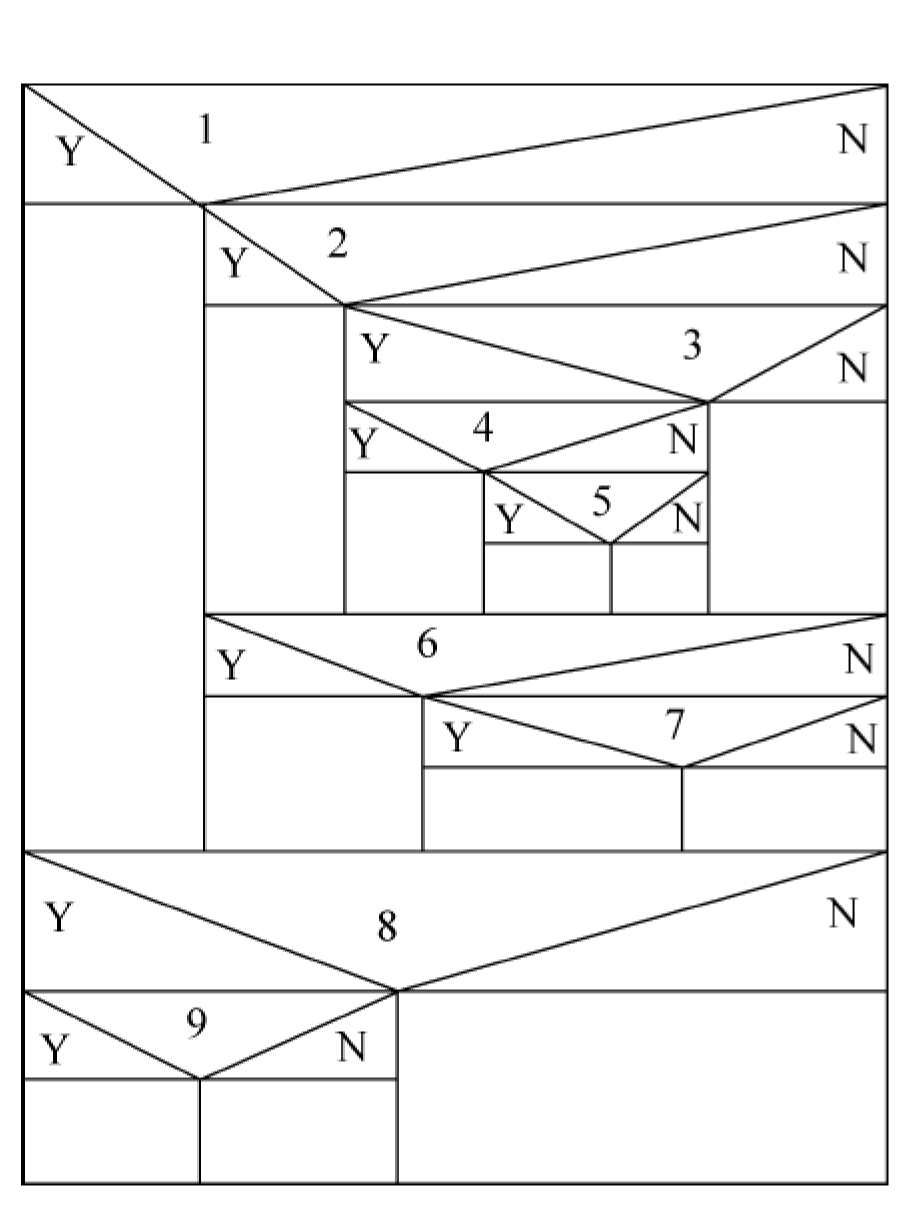


图 4.10　计算最少测试用例数实例

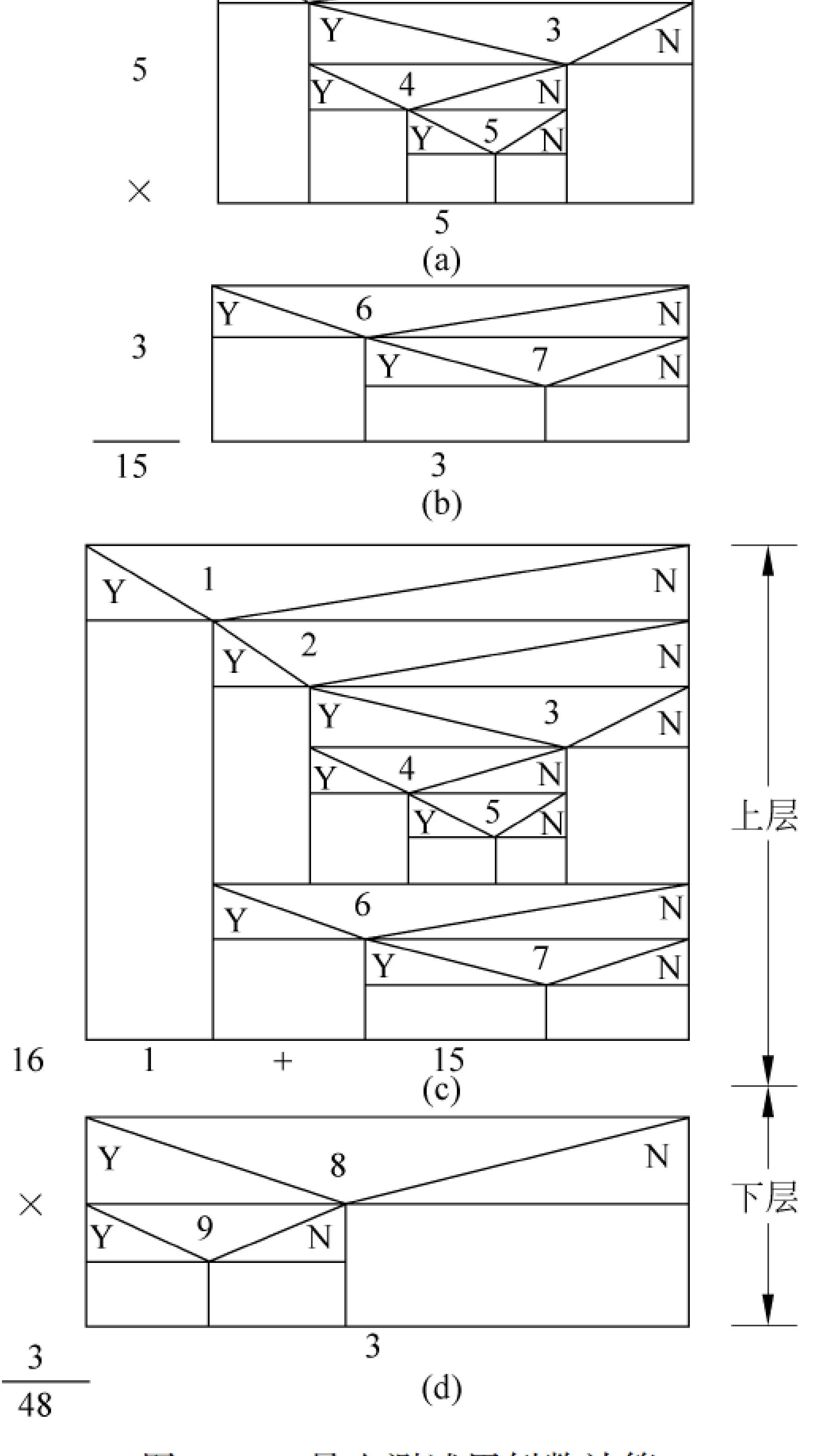


图 4.11　最少测试用例数计算

4.3　基本路径测试

基本路径测试也称独立路径测试，是在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出可执行的独立路径集合，从而设计出相应的测试用例。设计出的测试用例要保证被测程序的每条可执行的独立路径至少被执行一次。基本路径测试的具体步骤为：

- 导出程序流程图的拓扑结构——流图(控制流图)。
- 计算流图的环路复杂度。借助环路复杂度可以确定程序基本路径集合中的基路径条数。
- 利用"主路径＋转向"的策略确定基路径集合。首先从流图的入口结点出发，到出口结点结束，寻找一条包含尽可能多分支结点的路径记录下来作为主路径；然后以主路径为基础，依次对各个分支结点转向到另外一条未执行分支以产生新的路径，但是注意每次只改变一个分支结点的路径选择。当所有分支结点的可选路径都进行了转向覆盖后，这时得到的路径集就是基路径集合。
- 剔除不可行路径，补充其他重要的路径。如果各个分支条件判定变量存在前后依赖关系，则可能导致前面得到的基路径集合中存在不可行路径。因此需要对基路径集合进行进一步分析，找出不可行路径并将其剔除，然后补充路径以覆盖由于剔除不可行路径而遗漏的路径。
- 根据路径集合确定测试用例，填入测试数据。

例如，根据图 4.2 中所示控制流图，可以得到其基本路径集合如下：

path1：1,2,3,6,8,9,10,1,11

path2：1,11

path3：1,2,3,4,5,10,1,11

path4：1,2,3,6,7,9,10,1,11

该例中没有结出具体的程序代码，所以也就不需要进行不可行路径的判断。

由于测试用例要完成某条程序路径的执行，因此测试用例和测试用例所执行的程序路径之间有着非常明确的关系。

在路径测试中，最关键的问题仍然是如何设计测试用例，使之能够避免测试的盲目性，又能有较高的测试效率。一般有以下 3 个途径可得到测试用例。

(1) 通过非路径分析得到测试用例。

测试人员凭经验设计测试用例或由应用系统本身提供测试用例。在使用这些测试用例执行被测程序后，一些路径就被检测过了。

(2) 对未测试的路径生成相应的测试用例。

枚举被测程序所有可能的独立路径，并与前面已测试过的路径相比，便可得知哪些路径还没有被测试过，针对这些路径生成测试用例，进而完成对它们的测试。

(3) 生成指定路径的测试用例。

根据指定的路径，生成相应的测试用例。

按以上方法实施测试，原则上是可以做到路径覆盖的，原因如下。

- 对程序中的循环作了如上限制以后，程序路径的数量是有限的。
- 程序的路径可经枚举全部得到。
- 完成若干个测试用例后，对所测路径、未测路径是知道的。
- 在指出要测试的路径以后，可以自动生成相应的测试用例。

4.4 循环测试

循环是绝大多数软件算法的基础，但是，在测试软件时却往往未对循环结构进行足够的测试。

循环测试是一种白盒测试技术，它专注于测试循环结构的有效性。在结构化的程序中通常只有 3 种循环，即简单循环、串接循环和嵌套循环，如图 4.12 所示。下面讨论这 3 种循环的测试方法。

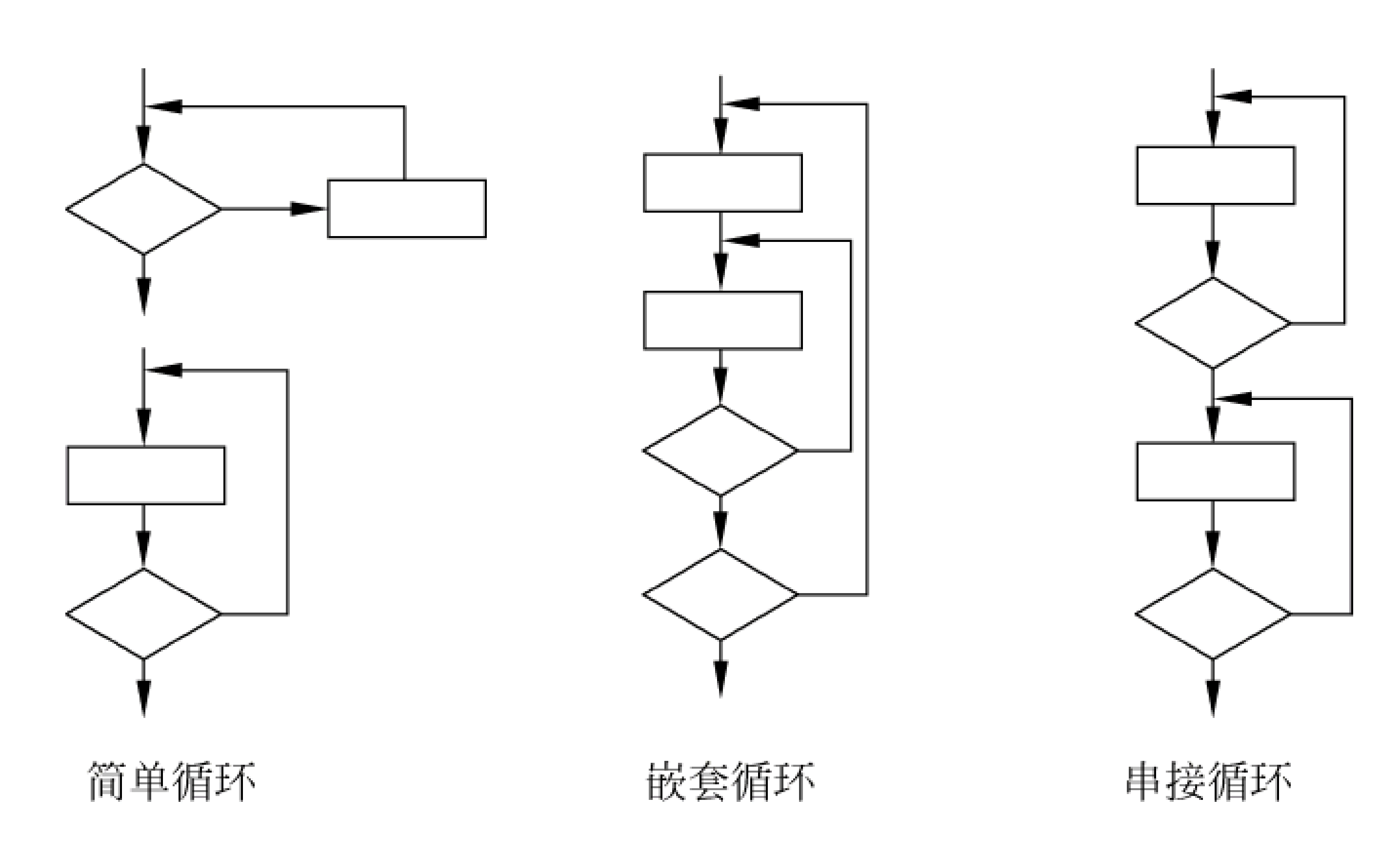


图 4.12　3 种循环

1. 简单循环

应该使用下列测试集来测试简单循环，其中，n 是允许通过循环的最大次数。

- 跳过循环。
- 只通过循环一次。
- 通过循环两次。
- 通过循环 m 次，其中 $m<n-1$。
- 通过循环 $n-1, n, n+1$ 次。

2. 嵌套循环

如果把简单循环的测试方法直接应用到嵌套循环，可能的测试数就会随嵌套层数的增加按几何级数增长，这会导致不切实际的测试数目。B. Beizer 提出了一种能减少测试数的方法：从最内层循环开始测试，把所有其他循环都设置为最小值；对最内层循环使用简单循环测试方法，而使外层循环的迭代参数(例如，循环计数器)取最小值，并为越界值或非法值增加一些额外的测试；由内向外，对下一个循环进行测试，但保持所有其他外层循环为最小值，其他嵌套循环为“典型”值；继续进行下去，直到测试完所有循环。

3. 串接循环

如果串接循环的各个循环都彼此独立，则可以使用前述的测试简单循环的方法来测试串接循环。但是，如果两个循环串接，而且第一个循环的循环计数器值是第二个循环的初始值，则这两个循环并不是独立的。当循环不独立时，建议使用测试嵌套循环的方法来测试串接循环。

4.5 面向对象的白盒测试

对面向对象软件的类测试相当于传统软件的单元测试。但与传统软件的单元测试不同的是，传统单元测试往往关注模块的算法细节和模块接口间流动的恶数据，而面向对象软件的类测试是由封装在类中的操作和类的状态行为所驱动的。

类测试一般有两种主要的方式：功能性测试和结构性测试，即对应于传统结构化软件的黑盒测试和白盒测试。

功能性测试以类的规格说明为基础，它主要检查类是否符合其规格说明的要求。例如，对于 Stack 类，即检查它的操作是否满足 LIFO 规则；结构性测试则从程序出发，它需要考虑其中的代码是否正确，同样是 Stack 类，就要检查其中代码是否正确且至少执行过一次。

结构性测试是对类中的方法进行测试，它把类作为一个单元来进行测试。测试分为两层：第一层考虑类中各独立方法的代码；第二层考虑方法之间的相互作用。

每个方法的测试要求能针对其所有的输入情况，但这样还不够，只有对这些方法之间的接口也做同样测试，才能认为测试是完整的。对于一个类的测试要保证类在其状态的代表集上能够正确工作，构造函数的参数选择以及消息序列的选择都要满足这一准则。因此，在这两个不同的测试层次上应分别做到以下两点。

(1) 方法的单独测试。

结构性测试的第一层是考虑各独立的方法，这可以与面向过程的测试采用同样的方法，两者之间最大的差别在于方法改变了它所在实例的状态，这就要取得隐藏的状态信息来估算测试的结果，传给其他对象的消息被忽略，而以桩来代替，并根据所传的消息返回相应的值，测试数据要求能完全覆盖类中代码，可以用传统的测试技术来获取。

(2) 方法的综合测试。

第二层要考虑一个方法调用本对象类中的其他方法和从一个类向其他类发送信息的情况。单独测试一个方法时，只考虑其本身执行的情况，而没有考虑动作的顺序问题，测试用例中加入了激发这些调用的信息，以检查它们是否正确运行了。对于同一类中方法之间的调用，一般只需要极少甚至不用附加数据，因为方法都是对类进行存取，故这一类测试的准则是要求遍历类的所有主要状态。

4.6 其他白盒测试方法简介

1. 域测试

域测试(domain testing)是一种基于程序结构的测试方法。Howden 把程序中出现的错误分为域错误、计算型错误和丢失路径错误 3 种。这是相对于执行程序的路径来说的。

我们知道，每条执行路径对应于输入域的一类情况，是程序的一个子计算。如果程序的控制流有错误，对于某一特定的输入可能执行的是一条错误路径，这种错误称为路径错误，也叫做域错误。如果对于特定输入执行的是正确路径，但由于赋值语句的错误致使输出结果不正确，则称此为计算型错误。另外一类错误是丢失路径错误，它是由于程序中某处少了一个判定谓词而引起的。域测试是主要针对域错误进行的程序测试。

域测试的“域”是指程序的输入空间。域测试方法基于对输入空间的分析。自然，任何一个被测程序都有一个输入空间。测试的理想结果就是检验输入空间中的每一个输入元素是否都产生正确的结果。而输入空间又可分为不同的子空间，每一子空间对应一种不同的计算。在考察被测试程序的结构以后，会发现子空间的划分是由程序中分支语句中的谓词决定的。输入空间的一个元素，经过程序中某些特定语句的执行而结束(当然也可能出现无限循环而无出口)，那都是满足了这些特定语句被执行所要求的条件。

域测试正是在分析输入域的基础上，选择适当的测试点以后进行测试的。域测试有两个致命的弱点：一是为进行域测试对程序提出的限制过多，二是当程序存在很多路径时，所需的测试点也很多。

2. 符号测试

符号测试的基本思想是允许程序的输入不仅仅是具体的数值数据，而且包括符号值，这一方法也因此而得名。这里所说的符号值可以是基本符号变量值，也可以是这些符号变量值的一个表达式。这样，在执行程序过程中以符号的计算代替了普通测试执行中对测试用例的数值计算，所得到的结果自然是符号公式或是符号谓词。更明确地说，普通测试执行的是算术运算，符号测试则是执行代数运算。因此符号测试可以认为是普通测试的扩充。

符号测试可以看作是程序测试和程序验证的一个折中方法。一方面，它沿用了传统的程序测试方法，通过运行被测程序来验证它的可靠性。另一方面，由于一次符号测试的结果代表了一大类普通测试的运行结果，实际上是证明了程序接受此类输入，所得输出是正确的还是错误的。最为理想的情况是，程序中仅有有限的几条执行路径。如果对这有限的几条路径都完成了符号测试，就能较有把握地确认程序的正确性。

从符号测试方法使用来看，问题的关键在于开发出功能更强，能够处理符号运算的编译器和解释器。

目前符号测试存在以下一些未得到圆满解决的问题。

(1) 分支问题。

当采用符号执行方法进行到某一分支点处，分支谓词是符号表达式，这种情况下通常无法决定谓词的取值，也就不能决定分支的走向，需要测试人员做人工干预，或是执行树的方法进行下去。如果程序中有循环，而循环次数又决定于输入变量，那就无法确定循环的次数。

(2) 二义性问题。

数据项的符号值可能是有二义性的。这种情况通常出现在带有数组的程序中，看以下的程序段：

```
...
X(I) = 2 + A
X(J) = 3
C = X(I)
...
```

如果 I＝J，则 C＝3，否则 C＝2＋A。但由于使用符号值运算，这时无法知道 I 是否等于 J。

(3) 大程序问题。

符号测试中总是要处理符号表达式。随着符号执行的继续，一些变量的符号表达式会越来越庞大。特别是当符号执行树如果很大，分支点很多，路径条件本身变成一个非常长的合取式。如果能够有办法将其化简，自然会带来很大好处。但如果找不到化简的办法，那么符号测试的时间和运行空间将大幅度的增长，甚至使整个问题难以解决。

3. Z 路径覆盖

分析程序中的路径是指：检验程序从入口开始，执行过程中经历的各个语句，直到出口。这是白盒测试最为典型的问题。着眼于路径分析的测试可称为路径测试。完成路径测试的理想情况是做到路径覆盖。对于比较简单的小程序实现路径覆盖是可能做到的。但是如果程序中出现多个判断和多个循环，可能的路径数目将会急剧增长，达到天文数字，以致不可能实现路径覆盖。

为了解决这一问题，必须舍掉一些次要因素，对循环机制进行简化，从而极大地减少路径的数量，使得覆盖这些有限的路径成为可能，称简化循环意义下的路径覆盖为 Z 路径覆盖。

这里所说的对循环化简是指限制循环的次数。无论循环的形式和实际执行循环体的次数多少，只考虑循环一次和零次两种情况，即只考虑执行时进入循环体一次和跳过循环体这两种情况。图 4.13(a)和(b)表示了两种最典型的循环控制结构。前者先作判断，循环体 B 可能执行(假定只执行一次)，也可能不执行，这就如同图 4.13(c)所表示的条件选择结构一样。后者先执行循环体 B(假定也只执行一次)，再经判断转出，其效果与图 4.13(c)中给出的条件选择结构只执行右分支的效果一样。

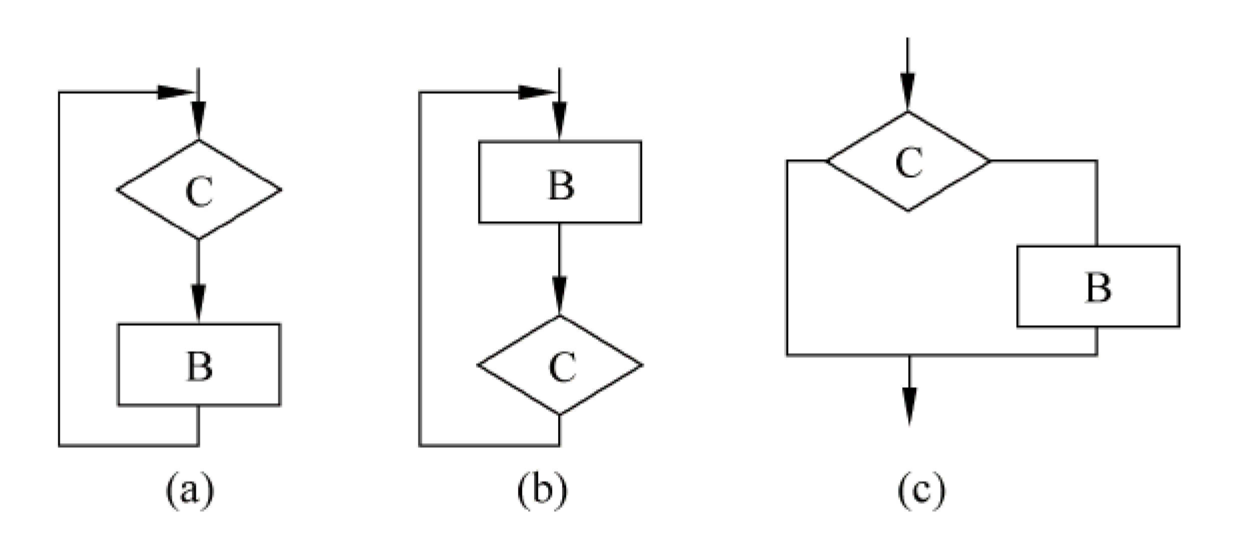


图 4.13　循环结构简化成选择结构

对于程序中的所有路径可以用路径树来表示。当得到某一程序的路径树后，可通过遍历路径树得到所有的路径。当得到所有的路径后，生成每个路径的测试用例，就可以做到 Z 路径覆盖测试。

练 习 题

1. 简述白盒测试用例的设计方法，并进行分析总结。

2. 分析归纳逻辑覆盖的各种策略，并比较每种覆盖的特点，分析在怎样的情况下采用何种覆盖方式。

3. 对如图 4.14 所示程序段进行语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖方法进行测试用例设计。

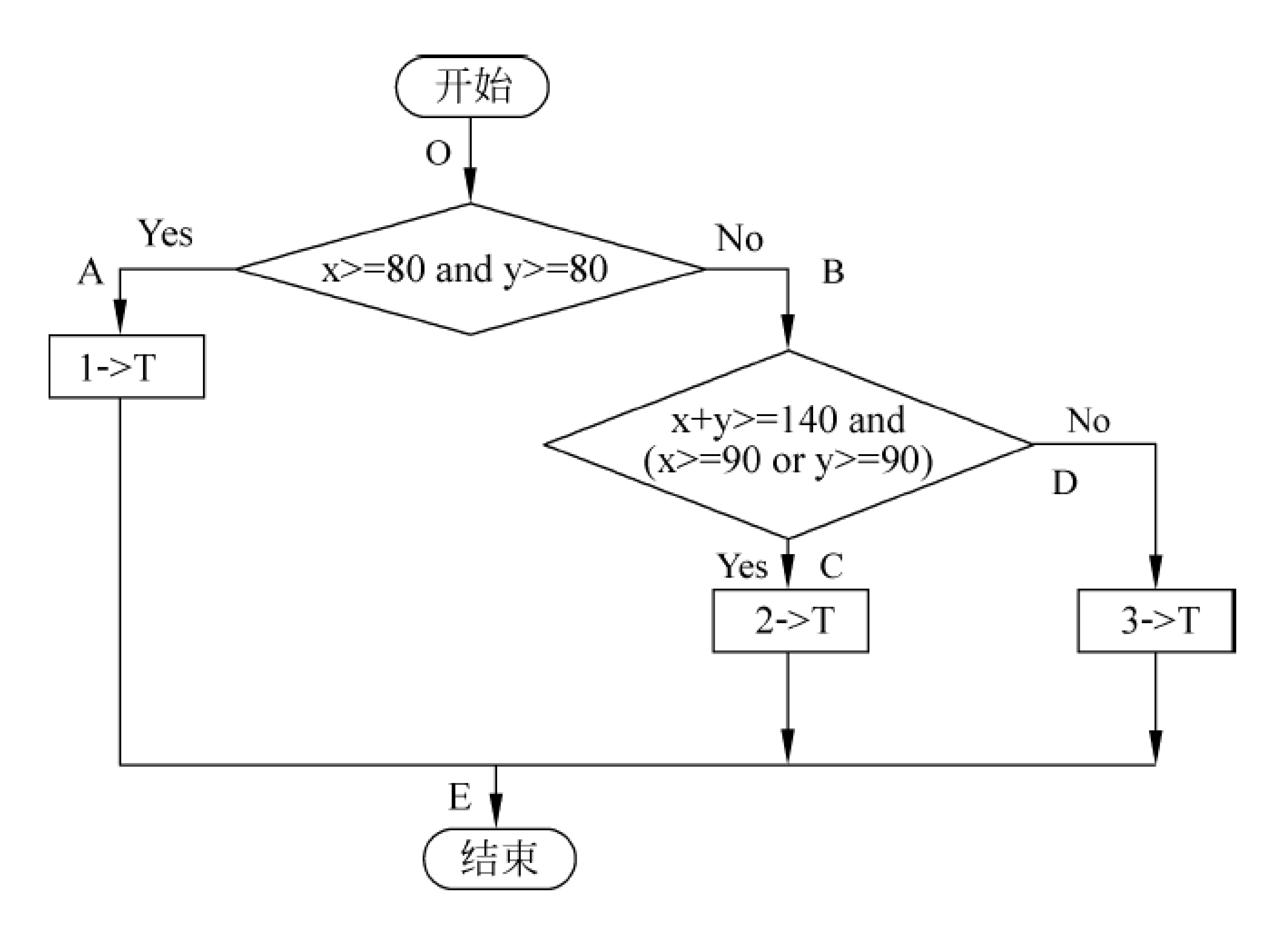


图 4.14 练习题 3

4. 请将下述语句按照各种覆盖方法设计测试用例。

```
if(a>2 && b<3 &&(c>4 || d<5))
{
   statement;
}
else
{
   statement;
}
```

5. 针对 test 函数按照基本路径测试方法设计测试用例。

```
int Test(int i_count,int i_flag)
    {
      int i_temp = 0;
      while (i_count>0)
      {
        if (0 == i_flag)
        {
          i_temp = i_count + 100;
          break;
```

```
        }
        else
        {
          if (1 == i_flag)
          {
            i_temp = i_temp + 10;
          }
          else
          {
           i_temp = i_temp + 20;
          }
         }
        i_count-- ;
      }
      return i_temp;
    }
```

6. 对如图 4.15 所示的 N-S 图，计算所需的最少测试用例数。

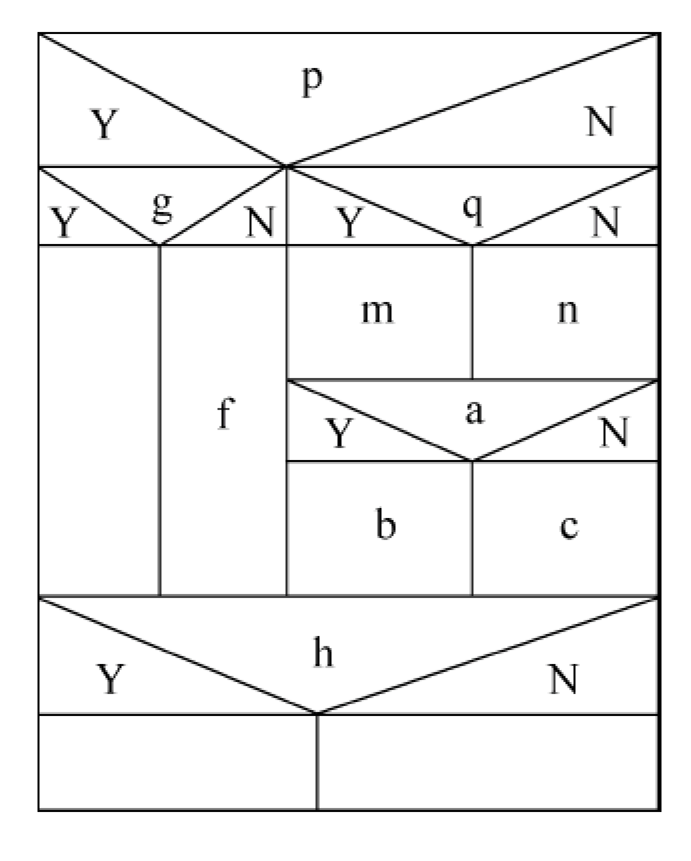


图 4.15　练习题 6

第5章 软件测试管理及自动化测试基础

5.1 软件测试自动化基础

软件测试是一项艰苦的工作,需要投入大量的时间和精力,据统计,软件测试会占用整个开发时间的40%。一些可靠性要求非常高的软件,测试时间甚至占到总开发时间的60%。但是软件测试具有一定的重复性,软件在发布之前要进行几轮测试。在测试后期所进行的回归测试中大部分测试工作是重复的,回归测试就是要验证已经实现的大部分功能。这种情况下,代码修改很少,针对代码变化所做的测试相对较少。而为了覆盖代码改动所造成的影响需要进行大量的测试,虽然这种测试找到软件缺陷的可能性小,效率比较低,但又是必要的。此后,软件不断升级,所要做的测试重复性也很高,所有这些因素驱动着软件测试自动化的产生和发展。

5.1.1 自动化测试含义

自动化测试是相对于手工测试而存在的,主要是使用软件工具来代替手工进行的一系列动作,具有良好的可操作性、可重复性和高效率等特点。自动化测试的目的是减轻手工测试的工作量,以达到节约资源(包括人力、物力等),保证软件质量,缩短测试周期的效果,是软件测试中提高测试效率、覆盖率和可靠性的重要测试手段。也可以说,测试自动化是软件测试不可分割的一部分。

自动化测试将毫无差错地以同一方式多次运行同一测试。但是自动化测试不会执行与脚本编写的内容不一样的行为。正因为如此,自动化测试通常被看成为一系列的回归测试,只能捕获被引入原来工作代码的缺陷。不过事情也会出现例外,例如大型数据数组循环输入。但是可以肯定自动化测试大都属于回归测试的范畴。

5.1.2 自动化测试意义

测试人员在进行手工测试时,具有创造性,可以举一反三,从一个测试用例想到另外一个测试用例,特别是可以考虑到测试用例没有覆盖的一些特殊的或边界的情况。同时,对于那些复杂的逻辑判断、界面是否友好,手工测试具有明显的优势。但是手工测试在某些测试方面,可能还存在着一定的局限性,例如:通过手工测试无法做到覆盖所有代码路径;简单的功能性测试用例在每一轮测试中都不能少,而且具有一定的机械性、重复性,其工作量往往较大,却无法体现手工测试的优越性;在系统负载、性能测试时,需要模拟大量数据或大量并发用户等各种应用场合,很难通过手工测试来进行。

由于手工测试的局限性,软件测试借助软件工具向自动化测试方向发展就显得极为必

要。通过自动化测试，可以解决上述手工测试的局限性，带来以下好处。

1. 提高测试效率

手工测试是一个劳动密集型的工作，并且容易出错。引入自动测试能够用更有效、可重复的自动化测试环境代替繁琐的手工测试活动，而且能在更少的时间内完成更多的测试工作，从而提高了测试工程师的工作效率。

2. 降低对软件新版本进行回归测试的开销

对于现代软件的迭代增量开发，每一个新版本大部分功能和界面都和上一个版本相似或完全相同，这时要对新版本再次进行已有的测试，这部分工作多为重复工作，特别适合使用自动化测试来完成，从而减小回归测试的开销。

3. 完成手工测试不能或难以完成的测试

对于一些非功能性方面的测试，如压力测试、并发测试、大数据量测试、崩溃性测试等，这些测试用手工测试是很难，甚至是不可能完成的。但自动化测试能方便地执行这些测试，比如并发测试，使用自动化测试工具就可以模拟来自多方的并发操作。

4. 具有一致性和可重复性

由于每次自动化测试运行的脚本是相同的，所以可以进行重复的测试，使得每次执行的测试具有一致性，手工测试则很难做到这点。

5. 更好地利用资源

将繁琐的测试任务自动化，可以使测试人员解脱出来，将精力更多地投入到测试案例的设计和必要的手工测试当中。并且理想的自动化测试能够按计划完全自动地运行，使得完全可以利用非工作时间执行自动测试。

6. 降低风险，增加软件信任度

自动化测试能通过较少的开销获得更彻底的测试效果，从而更好地提高了软件产品的质量。

5.1.3 自动化测试局限性

当然，自动化测试也并非万能，它所完成的测试功能也是有限的，不可能也没有必要取代手工测试来完成所有的测试任务。以下几点是自动化测试的不足。

1. 软件自动化测试可能降低测试的效率。当测试人员只需要进行很少量的测试，而且这种测试在以后的重用性很低时，花大量的精力和时间去进行自动化的结果往往是得不偿失。因为自动化的收益一般要在很多次重复使用中才能体现出来。

2. 测试人员期望自动测试发现大量的错误。测试首次运行时，可能发现大量错误。但当进行过多次测试后，发现错误的几率会相对较小，除非对软件进行了修改或在不同的环境下运行。

3. 如果缺乏测试经验，测试的组织差、文档少或不一致，则自动化测试的效果比较差。

4. 技术问题。毫无疑问商用软件自动测试工具是软件产品。作为第三方的技术产品，如果不具备解决问题的能力和技术支持或者产品适应环境变化的能力不强，将使得软件自动化工具的作用大大降低。

因此，对软件自动化测试应该有正确的认识，它并不能完全代替手工测试。不要期望仅仅通过自动化测试就能提高测试的质量，如果测试人员缺少测试的技能，那么测试也可能会失败。

5.1.4 测试工具

测试工具可以从以下两个不同的方面去分类。

根据测试方法不同,分为白盒测试工具和黑盒测试工具。

根据测试的对象和目的,分为单元测试工具、功能测试工具、负载测试工具、性能测试工具和测试管理工具等。

1. 白盒测试工具

白盒测试工具是针对程序代码、程序结构、对象属性、类层次等进行测试,测试中发现的缺陷可以定位到代码行、对象或变量级。根据测试工具原理的不同,又可以分为静态测试工具和动态测试工具。

静态测试工具对代码进行语法扫描,找出不符合编码规范的地方,根据某种质量模型评价代码的质量,生成系统的调用关系图等。它直接对代码进行分析,不需要运行代码,也不需要对代码编译链接、生成可执行文件。

动态测试工具与静态测试工具不同,需要实际运行被测系统,并设置断点,向代码生成的可执行文件中插入一些监测代码,掌握断点这一时刻程序运行数据(对象属性、变量的值等)。

单元测试工具多属于白盒测试工具。

2. 黑盒测试工具

黑盒测试工具适用于系统功能测试和性能测试,包括功能测试工具、负载测试工具、性能测试工具等。黑盒测试工具的一般原理是利用脚本的录制(Record)/回放(Playback),模拟用户的操作,然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量,在迭代开发的过程中,能够很好地进行回归测试。

3. 其他测试工具

在上述两类测试工具之外还有测试管理工具,这类工具负责对测试计划、测试用例、测试实施进行管理,对产品缺陷跟踪管理、产品特性管理等。

除了上述的测试工具外,还有一些专用的测试工具,例如,针对数据库测试的TestBytes,对应用性能进行优化的 EcoScope 等工具。

5.2 软件测试管理

随着计算机硬件成本的不断下降,软件在整个计算机系统的成本占有越来越高的比例,如何提高软件质量是整个计算机软件行业的重大课题。软件测试作为软件开发的一个重要环节,越来越受重视。为了尽可能多地找出程序中的错误,保证软件产品的质量,就需要对软件测试进行有效的管理,确保测试工作顺利进行。

实践证明,对软件进行测试管理可及早发现错误,避免大规模返工,降低软件开发费用。为确保最终软件质量符合要求,必须进行测试与管理。对于不同企业的不同类产品、不同企业的同一类产品或同一企业的不同类产品,其各阶段结果的形式与内容都会有很大的不同。所以,对于软件测试管理除了要考虑测试管理开始的时间、测试管理的执行者、测试管理技术如何有助于防止错误的发生、测试管理活动如何被集成到软件过程的模型中之外,还必须

在测试之前，制定详细的测试管理计划，充分实现软件测试管理的主要功能，缩短测试管理的周期。

5.2.1 软件测试管理计划

一个成功的测试开始于一个全面的测试管理计划。因此，在每次测试之前应做好详细的测试管理计划。

首先应该了解被测对象的基本信息，选择测试的标准级别，明确测试管理计划标识和测试管理项。在定义了被测对象的测试管理目标、范围后必须确定测试管理所使用的方法，即提供技术性的测试管理策略和测试管理过程。在测试管理计划中，管理者应该全面了解被测试对象的系统方法、语言特征、结构特点、操作方法和特殊需求等，以便确定必要的测试环境，包括测试硬件、软件及测试环境的建立等。而且，在测试管理计划中还应该制订一份详细的进度计划，如：测试管理的开始段、中间段、结束段及测试管理过程每个部分的负责人等。

由于任何一个软件不可能没有缺陷、系统运行时不出现故障，所以在测试管理计划中还必须考虑到一些意外情况，也就是说，当问题发生时应如何处理。因为测试管理具有一定难度，所以对测试管理者应进行必要的测试设计、工具、环境等的培训。

最后，还必须确定认可和审议测试管理计划的负责人员。

5.2.2 软件测试管理过程

一般来讲，由对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。

在实现组将所开发的程序经验证后，提交测试组，由测试负责人组织测试，测试一般可按下列方式组织。

1. 首先，测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试前的准备工作。

2. 为了保证测试的质量，将测试过程分成几个阶段，即：代码审查、单元测试、集成测试、确认测试和系统测试。

① 代码审查。

代码审查是一组人通过阅读、讨论和争议对程序进行静态分析的过程。审查小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开代码审查会议，程序员逐句讲解程序的逻辑，并展开热烈的讨论甚至争议，以揭示错误的关键所在。实践表明，程序员在讲解过程中能发现许多自己原来没有发现的错误，而讨论和争议则进一步促使了问题的暴露。

② 单元测试。

单元测试集中在检查软件设计的最小单位——模块上，通过测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。

③ 集成测试。

集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。如，数据穿过接口时可能丢失；一个模块与另一个模块可能有由于疏忽的问题而

造成有害影响；把子功能组合起来可能不产生预期的主功能；个别看起来是可以接受的误差可能积累到不能接受的程度；全局数据结构可能有错误等。

④ 确认测试。

确认测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是确认测试的任务，即软件的功能和性能如同用户所期待的那样。

⑤ 系统测试。

软件开发完成以后，最终还要与系统中其他部分配套运行，进行系统测试。包括恢复测试、安全测试、强度测试和性能测试等。

在整个过程当中需要对测试过程中每个状态进行记录、跟踪和管理，并提供相关的分析和统计功能，生成和打印各种分析统计报表。通过对详细记录的分析，形成较为完整的软件测试管理文档，保障软件在开发过程中避免同样的错误再次发生，从而提高软件开发质量。

5.2.3 软件测试的人员组织

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对分析、设计和实现等各阶段所得到的结果，包括需求规格说明、设计规格说明及源程序都应进行软件测试。基于此，软件测试人员的组织应分以下阶段。

1. 软件的设计和实现都是基于需求分析规格说明进行

需求分析规格说明是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量，应对其进行严格的审查。审查小组通常由一名组长和若干成员组成，其成员包括系统分析员，软件开发管理者，软件设计、开发、测试人员和用户。

2. 设计评审

软件设计是将软件需求转换成软件表示的过程。主要描绘出系统结构、详细的处理过程和数据库模式。按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，同时利用关系数据库的规范化理论对数据库模式进行审查。评审小组由下列人员组成：组长一名，成员包括系统分析员、软件设计人员、测试负责人员各一名。

3. 软件测试

软件测试是软件质量保证的关键。软件测试在软件生存周期中横跨两个阶段。通常在编写出每一个模块之后，就对它进行必要的测试(称为单元测试)。编码与单元测试属于软件生存周期中的同一阶段。该阶段的测试工作，由编程组内部人员进行交叉测试(避免编程人员测试自己的程序)。这一阶段结束后，进入软件生存周期的测试阶段，对软件系统进行各种综合的测试。测试工作由专门的测试组完成，测试组设组长一名，负责整个测试的计划、组织工作。测试组的其他成员由具有一定的分析、设计和编程经验的专业人员组成，人数根据具体情况可多可少，一般3～5人为宜。

5.2.4 软件测试管理主要功能

1. 测试控制对象的编辑和管理

测试控制对象包括测试方案、测试案例、各案例的具体测试步骤、问题报告、测试结果报

告等，该部分主要是为各测试阶段的控制对象提供一个完善的编辑和管理环境。

2. 测试流程控制和管理

测试流程的控制和管理是基于科学的流程和具体的规范来实现的，并利用该流程和规范，严格约束和控制整个产品的测试周期，以确保产品的质量。整个过程避免了测试人员和开发设计人员之间面对面的交流，减少了以往测试和开发之间难免的摩擦和矛盾，提高了工作效率。

3. 统计分析和决策支持

在系统建立的测试数据库的基础上，进行合理的统计分析和数据挖掘，例如根据问题分布的模块、问题所属的性质、问题的解决情况等方面的统计分析使项目管理者全面了解产品开发的进度、产品开发的质量、产品开发中问题的聚集，为决策管理提供支持。例如，设计人员在遇到问题时可以到案例库中查找类似问题的解决办法等。

5.2.5 软件测试管理实施

任何程序，无论大小，都可能会有错误发生。每一个新版本都需要进行新特性的测试和其他特性的一些回归测试。所以软件测试管理具有周期性。

测试管理人员在接受一个测试管理任务后，除了要制定周密的测试管理计划。还要进行测试方案管理；并且对测试人员所做的测试活动予以记录，做好测试流程的管理。同时，对发现的缺陷予以标识，一方面反馈给提交测试的人员；另一方面将存在的问题和缺陷存入案例库，直至测试通过。

软件测试是一个完整的体系，主要由测试规划、测试设计、测试实施、资源管理等相互关联、相互作用的过程构成。软件测试管理系统可以对各过程进行全面控制，具体的实现过程如下。

（1）按照国际质量管理标准，建立适合本公司的软件测试管理体系，以提高公司开发的软件质量，并降低软件开发及维护的成本。

（2）建立、监测和分析软件测试过程，以有效地控制、管理和改进软件测试过程。监测软件质量，从而确定交付或发布软件的时间。

（3）制定合理的软件测试管理计划，设计有效的测试案例集，以尽可能发现软件缺陷，并组织、管理和应用庞大的测试案例集。

（4）在软件测试管理过程中，管理者、程序员、测试员（含有关客户人员）协同工作，及时解决发现的软件问题。

（5）对于软件测试中发现的大量的软件缺陷，进行合理的分类以分清轻重缓急。同时进行原因分析，并做好相应的记录、跟踪和管理工作。

（6）建立一套完整的文档资料管理体系。因为软件测试管理很大程度上是通过对文档资料的管理来实现。软件测试每个阶段的文档资料都是以后阶段的基础，又是对前面阶段的复审。

5.2.6 软件测试管理工具简介

在软件测试管理周期中，为了便于对制定测试方案、编写测试案例和测试步骤等各个阶段进行有效的控制和管理，为了提高软件开发和产品测试的管理水平，保证软件产品质量，

软件测试管理工具是非常重要的手段。在此介绍以下一些比较流行的软件测试管理工具。

（1）软件测试管理系统（TMS）：TMS 管理功能全面，对测试流程的设计科学、规范、合理。系统的开发是在充分借鉴 Microsoft、Nortel 等国际知名大公司在测试领域尤其是测试流程管理方面的经验，参考 SQA Manager 等国外知名测试管理软件，并结合开发人员在业界的经验和对国内软件开发现状的把握等基础上开发而成，非常贴近国内用户的需求。它具有很强大的测试案例、测试步骤的编辑和管理功能，问题（缺陷）的跟踪处理功能，所有输出结果自动生成 Word 文档的功能，同时有强大的统计分析、决策支持能力。该系统技术实现上采用 Web/Browser 开发模式，使用维护方便，具有良好的性价比。

（2）测试管理工具（Test Management Workshop）：该系统定义了一个良好的表单归档机制，并且支持这些表单的交叉引用。通过关键项目文档中内建的关联设计，用户可以根据不同的线索追踪和调用相关的文档。同时，所有文档均被置于严格的安全控制之下，而且客户端支持浏览器方式操作。

（3）测试管理工具（Jactus Labs）：Jactus Labs 为了适应数以百计的用户，有一个中心数据储存库，所有的用户可以共享并存取主要的信息——测试脚本、缺陷及报告书。该测试管理工具把测试计划、测试执行和缺陷跟踪三者有机地结合在一起，同时为了更多的灵活性还采用了开放式测试结构（Open Test Architecture，OTA）。它利用 Microsoft Access 数据库缩小安装，并利用符合行业标准的关系数据库（包括 Oracle、Microsoft SQL Server 和 Sybase）来扩大安装测试管理工具。对于每一件测试案例，它都会列出用户操作的顺序、案例描述、状态和预期的结果。这些信息都可以逐步填在一张校验表里并被记录在所有测试案例文件中，从而使测试过程更合理、统一。

（4）软件测试管理系统（i-Test）：i-Test 采用 B/S 结构，可以安装在 Web 服务器上。项目有关人员都可以在不同地点通过 Internet 同时登录和使用，协同完成软件测试，减少了为集中人员而出差所产生的费用。同时，该系统提供相应的自动化功能，可高效编写、查询和引用测试用例，快速填写、修改和查询软件缺陷报告，并提供相关的分析和统计功能，生成和打印各种分析统计报表。

这些软件测试管理工具可以为企业商业系统提供全面的、综合的测试管理解决方案，并可以控制和管理所有的测试工作来确保测试是一个有组织的、规范文档化的和全面的测试活动。

练　习　题

1. 简述软件测试自动化的意义。
2. 在运用软件自动化测试时，应注意哪些缺点或事项？
3. 软件测试工具主要分为哪几大类？
4. 了解时下常用的自动化测试用具，并对这些工具进行针对性说明。
5. 简述软件测试管理过程。
6. 简述软件测试管理的主要功能。
7. 试选择一个小型的应用系统，做功能测试计划并设计测试用例。

第 6 章 WinRunner 测试工具

6.1 功能测试工具简介

软件功能测试是典型的黑盒测试,主要检查实际软件的功能是否符合用户的需求。软件是由很多功能组成的,而这些功能是源自人们生活中各方面的需求。有了这些需求,才有相应的软件,反过来,软件编好之后,要对它进行功能测试,看看这些功能是否满足了用户的实际需求。

通过第 5 章的学习,知道如果合理地运用软件测试工具,可以使测试工作事半功倍,达到出其不意的效果。目前市场上专业开发软件测试工具的公司很多,例如 MI 公司、Rational 公司等。其中 MI 公司的 WinRunner、Rational 公司的 Robot 以及 Compuware 公司的 QARun 等都是属于功能测试工具,它们的基本原理都是通过录制和回放来实现自动化的功能测试,只是在具体实现形式上有所差别。

目前 MI 公司开发的软件测试工具在市场上处于主流地位,因此本章将重点学习 WinRunner 测试工具的基本使用方法。

6.2 WinRunner 简介

WinRunner(缩略为 WR,这里介绍的是 V8.2)是基于 MS Windows 的功能测试工具。WR 可以帮助测试人员自动处理从测试开发到测试执行的整个过程。测试人员可以创建可修改的、可复用的测试脚本,而且不用担心软件功能模块的变更。测试人员只需要让计算机自动执行这些脚本,就能轻而易举地发现软件中的错误,从而确保软件的质量。

6.2.1 运行

执行【开始】→【程序】→【WinRunner】→【WinRunner】命令,弹出图 6.1 所示的窗口。

在 WinRunner 启动时,可以选择支持 ActiveX Controls、PowerBuilder、Visual Basic 或 Web Test 的插件,如图 6.1 所示。其他插件需要单独向 MI 公司购买,建议不要同时载入所有的插件,不必要的插件可能会对录制或执行脚本造成问题。

将【Show on startup】前面的勾去掉,这个 Add-In Manager 窗口就不会在 WR 启动的时候出现。测试人员也可以在进入 WR 后通过【Tools】→【General Options】→【startup】命令设置是否在 WinRunner 启动时显示这个窗口以及等待的时间等。

单击【OK】按钮,弹出如图 6.2 所示的窗口。



图 6.1　WinRunner Add-in Manager 窗口



图 6.2　WinRunner 欢迎窗口

可以选择创建一个新的测试或打开一个现有的测试，也可以看 WinRunner 的快速演示。

选择创建新测试后进入主界面，如图 6.3 所示。

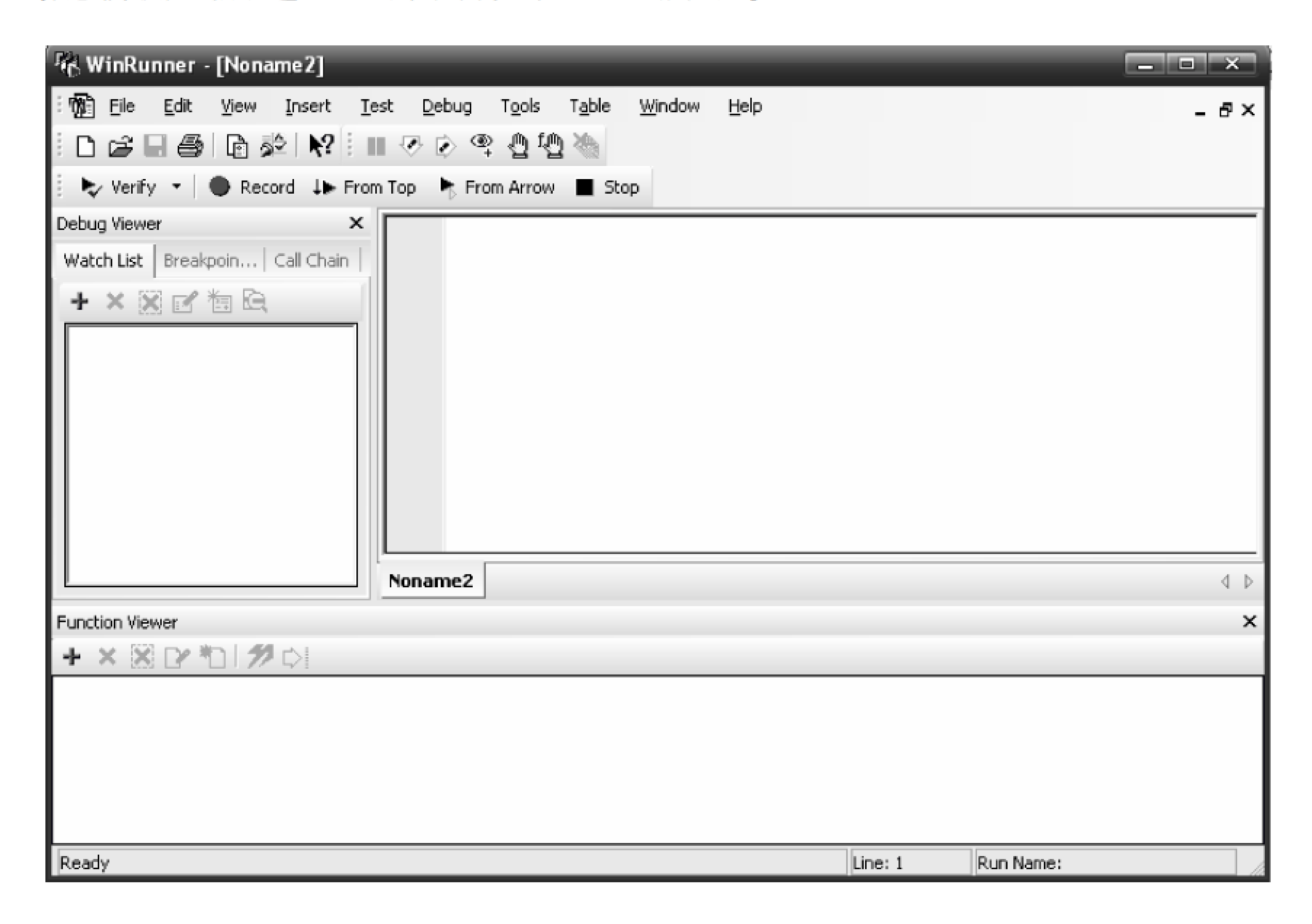
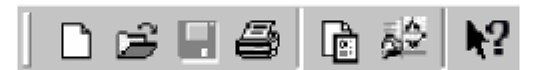


图 6.3　WinRunner 主界面

1. 档案(File)工具列

提供常用的按钮，如新增、开启、储存等。



：New(Ctrl+N)，建立新的测试脚本。

：Open(Ctrl+O)，开启旧的测试脚本。

：Save(Ctrl+S)，储存测试脚本。

：Print(Ctrl+P)，打印。

：Test Properties，设定测试脚本的属性。

：Test Results，检视测试脚本的测试结果。

：Help(Shift+F1)，说明文件。

2. 测试(Test)工具列

提供常用的按钮，如录制、执行、停止等。

Verify ▾ | Record | From Top | From Arrow | Stop

Verify ▾：Run Mode，设定执行模式，有 Verify、Debug、Update 共 3 种执行模式。

Record：Click to Record in Context Sensitive，开始以 Context Sensitive 模式录制。

From Top：Run from Top，从头开始执行。

From Arrow：Run from Arrow，从黄色小箭头处开始执行。

Stop：Stop，停止录制或执行。

3. 调试(Debug)工具列

提供除错时常用的按钮，如逐步执行、暂停、设定断点等。

：Pause，暂停。

：Step，逐步执行。

：Step Into，逐步执行并进入。

：Add Watch，新增监视变数。

：Toggle Breakpoint(F9)，设置断点。

：Break in Function(Ctrl+B)，设置函数中的中断点。

：Delete All Breakpoints，清除所有断点。

4. 使用者(User)工具列

提供让使用者自订常用的按钮。

：Click to Record in Context Sensitive，开始以 Context Sensitive 模式录制。

：Stop，停止录制或执行。

：Insert Function for Object/Window，在对象/窗口插入函数。

：Insert Function from Function Generator，从函数生成器中插入函数。

：GUI Checkpoint for Object/Window，为对象/窗口建立 GUI 检查点。

：GUI Checkpoint for Multiple Objects，为多个对象建立 GUI 检查点。

：Bitmap Checkpoint for Object/Window，为对象/窗口建立位图检查点。

：Bitmap Checkpoint for Screen Area，为屏幕区域建立位图检查点。

：Default Database Checkpoint，默认数据库检查点。

：Synchronization Point for Object/Window Property，为对象/窗口属性建立同步点。

：Synchronization Point for Object/Window Bitmap，为对象/窗口位图建立同步点。

：Synchronization Point for Screen Area Bitmap，为屏幕区域位图建立同步点。

：Get Text from Object/Window，从对象/窗口读取文本信息。

：Get Text from Screen Area，从屏幕区域读取文本信息。

6.2.2 测试模式

在进行录制脚本的时候，可以很快的录制出自动化操作的脚本。利用 WinRunner 可以记录测试人员的操作过程和在测试脚本语言中所体现的声明。在 WinRunner 中这些都是测试的脚本，其语言是被称作 TSL(Test Script Language)的一种类 C 语言。在开始测试之前，测试人员要为测试中主要的地方或重要的阶段选择适当的记录模式，根据不同的情况，它在录制脚本时包括两种测试的模式：环境判断模式(Context Sensitive mode)和模拟模式(Analog mode)。

1. 环境判断模式

该模式在测试中能记录测试人员绝大部分的操作过程。WinRunner 会确定测试人员每一个所单击的操作(包括窗口菜单、目录和按钮)和典型的操作任务。Context Sensitive 模式根据测试人员所选择的 GUI(图形用户界面)对象(例如窗口、目录还有按钮等)将用户对软件的操作动作录制下来，虽然它忽略了关于对象在屏幕上的物理位置，但是每次录制测试脚本的时候，TSL 将综合描述 GUI 对象在执行时发生在测试脚本里的动作。WinRunner 对每一个选择的对象写一个唯一的 GUI map 来进行描述。GUI map 在于把文件维护从测试脚本中分离。如果应用软件用户界面改变了，测试人员只要更新 GUI map，不必重新录制。WinRunner 模仿一个操作者移动鼠标指针、关闭应用软件、选择对象和加入键盘输入。WinRunner 在 GUI map 中读取与应用程序的 GUI 对象相匹配的逻辑名和物理描述，即使对象位置改变也能在窗口中定位。

在【Test】菜单中单击【Record-Context Sensitive】选项就可以实现环境判断模式，一般在使用中默认为该模式。

2. 模拟模式

这种模式记录鼠标单击、键盘输入和鼠标在二维平面上(x 轴、y 轴)的精确运动轨迹。执行测试时，WR 让鼠标根据轨迹运动。这种模式对于那些需要追踪鼠标运动的测试非常有用，例如画图软件。

在【Test】的菜单中单击【Record-Analog】选项后就可以实现模拟模式。

6.2.3 测试过程

WR 的测试过程分为创建 GUI map、创建测试、调试测试、执行测试、查看测试结果和报告发现的错误 6 个阶段。

1. 创建 GUI map

使用 RapidTest Script wizard(快速测试脚本指南)回顾软件用户界面，并系统地把每个 GUI 对象的描述添加到 GUI map 中。测试人员也可以在录制测试的时候，通过单击对象把对单个对象的描述添加到 GUI map 中。

注意：当使用 GUI map per test 模式时，测试人员可以跳过这一步骤。

2. 创建测试

测试人员可以通过录制、编程或两者同时使用的方式创建测试脚本。录制测试时，在测试人员需要检查软件反应的地方插入检查点(Checkpoint)。测试人员可以插入检查点来检查 GUI 对象、位图(Bitmap)和数据库。在这个过程中，WR 捕捉数据并作为期望结果(被测软件的期望反应)储存下来。

3. 调试测试

测试人员可以先在调试模式(Debug mode)下运行脚本。测试人员也可以设置中断点(Breakpoint)，监测变量，控制 WR 识别和隔离错误。调试结果被保存在调试文件夹(Debug folder)中，一旦调试结束就可以删除。

4. 执行测试

测试人员在检验模式(Verify mode)下测试被测软件。WR 在脚本运行中遇到检查点后，就把当前数据和前期捕捉的期望值进行比较。如果发现有不符合的，就记录下来作为实测结果。

5. 查看测试结果

测试是否成功由测试者来认定。每次测试结束，WR 会把结果显示在报告中。报告会详述测试执行过程中发生的所有主要事件，如检查点、错误信息、系统信息或用户信息。

如果在检查点有不符合被发现，测试者可以在 Test Results(测试结果)窗口查看预期结果和实测结果。如果是位图不符合，测试人员也可以查看显示预期值和实测结果之间差异的位图。

6. 报告发现的错误

如果由于测试中发现错误而造成测试运行失败，测试人员可以直接从测试结果(Test Results)窗口报告有关错误的信息。这些信息通过 E-mail 发送给测试经理(QA Manager)，用来跟踪这个错误直到被修复。

6.2.4 样本软件

本章例子是使用 WR 附带的 Flight Reservation(航班预订)软件。

1. 启动样本软件

样本软件启动方法是：选择【开始】→【程序】→【WinRunner】→【Sample Application】命令。该程序有两个版本 Flight 4A 和 Flight 4B。

2. 样本软件的多个版本

Flight 4A 这个版本是正常的软件，Flight 4B 有一些故意加入的错误。

如果 WR 中安装了 Visual Basic 支持，VB 版本的 Flight 1A 和 Flight 1B 将被安装到常规样本软件中。

3. 登录

使用任意用户名(长度至少 4 个字符)登录 Flight Reservation(航班预订)软件，密码为：Mercury。

4. WEB 版样品软件

WEB 版样品软件可以在 http://MercuryTours.mercuryinteractive.com 上下载；或通

过选取【开始】→【程序】→【WinRunner】→【Sample Application】→【Mercury Tours site】命令来访问。

6.2.5 测试套件

WR 和测试套件的其他工具一起提供整个测试流程的解决方案：测试计划、测试开发、GUI 测试、错误跟踪和多用户系统客户端负载测试。

1. TestDirector

TestDirector 是测试管理工具，帮助测试人员计划和组织测试活动。用 TD 可以创建手工和自动控制测试的数据库、建立测试流、执行测试、跟踪和报告错误。

2. LoadRunner

LoadRunner 是用于 Client/Server 结构软件(Browser/Server 结构也可以使用)的测试工具，可以模拟多个用户登录到一台服务器的情况。

6.3 GUI Map

一般的 Windows 应用程序，通常是由窗口、按钮、菜单等组成，在 WinRunner 中这些窗口、按钮、菜单等通称为 GUI 对象(GUI object)。WinRunner 会通过这些 GUI 对象的属性(physical properties)，如 class、label、width、height、handle 与 enabled 等，来识别 GUI 对象。WinRunner 只会记录最少但可组合成唯一的属性来辨识 GUI 对象。例如，当 WinRunner 识别一个 OK 按钮时，会记录这个按钮所属的窗口(如属于 Open 窗口中的 GUI 对象)、隶属的 class(如 push_button)、按钮的文字卷标(如 OK)来识别此按钮，至于如 width、height、handle 或其他属性就不会被使用到。

当 WR 运行测试时，它模拟一个真实的用户对软件的 GUI 对象用鼠标、键盘进行操作。因此，WR 必须学习软件的 GUI，学习 GUI 对象和对象的属性。测试人员可以用 GUI Spy 查看任意 GUI 对象的属性，了解 WR 是如何识别它们的。

WR 通过以下方式学习软件的 GUI。

(1) 使用 RapidTest Script wizard 学习软件每个窗体中所有 GUI 对象的属性。

(2) 通过录制脚本的方法学习被录制的那部分软件中所有的 GUI 对象的属性。

(3) 使用 GUI Map Editor 学习单个 GUI 对象、窗体或某个窗体中所有 GUI 对象的属性。

如果软件开发过程中 GUI 改变了，测试人员可以使用 GUI Map Editor 更新 GUI map。

在测试人员开始让 WR 学习软件的 GUI 之前，测试人员需要确认测试人员组织 GUI map 文件的方式，主要有以下两种方式。

(1) GUI Map per Test mode，为每个新建的测试创建一个新的 GUI map 文件。

(2) Global GUI map File mode，相关测试共享同一个 GUI map 文件。

6.3.1 GUI 对象属性的查看

WinRunner 提供一个工具叫 GUI Spy，可以用来检视某个 GUI 对象有哪些属性以及 WinRunner 以哪些属性来识别此 GUI 对象。

实例 1：

以 GUI Spy 检视 Flight Reservation 范例程序登录窗口的 GUI 对象。

1. 开启 Flight Reservation 范例程序

执行【开始】→【程序】→【WinRunner】→【Sample Applications】→【Flight 4A】命令，登录窗口会开启，如图 6.4 所示。

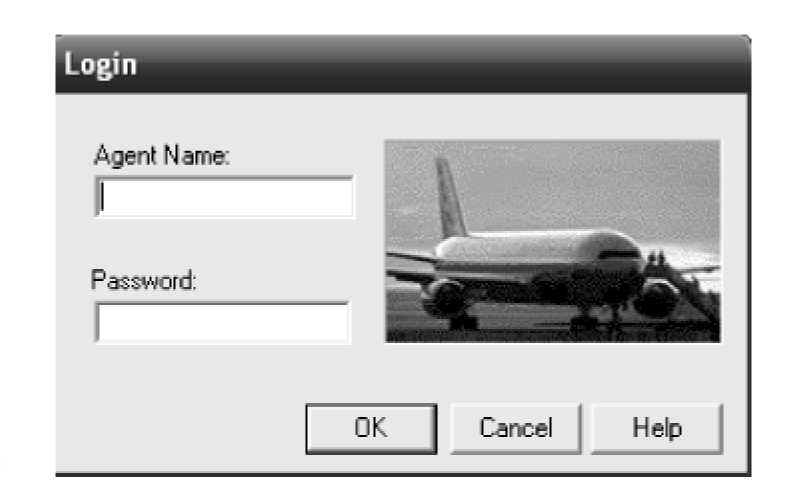


图 6.4 Flight 4A 登录窗口

2. 开启 WinRunner

执行【开始】→【程序】→【WinRunner】命令，如果是第一次执行 WinRunner，会开启欢迎窗口，则选择【New Test】选项；如果没有开启欢迎窗口，则选择【File】→【New】命令。

3. 开启 GUI Spy

选择【Tools】→【GUI Spy】命令开启 GUI Spy，选择【Hide WinRunner】复选框，如图 6.5 所示。

4. 检视 WinRunner 用来识别【OK】按钮的属性

在 GUI Spy 单击【Spy】按钮，WinRunner 会缩到最小，这时可以看到 Flight Reservation 的登录窗口，将鼠标移动到登录窗口上，可以看到被鼠标指到的 GUI 对象会有个外框在闪动，同时 GUI Spy 也会显示此 GUI 对象的属性。

将鼠标移到【OK】按钮上，然后单击左边的【Ctrl+F3】项，会弹出 Spy 模式，GUI Spy 中会显示【OK】按钮的属性，如图 6.6 所示。

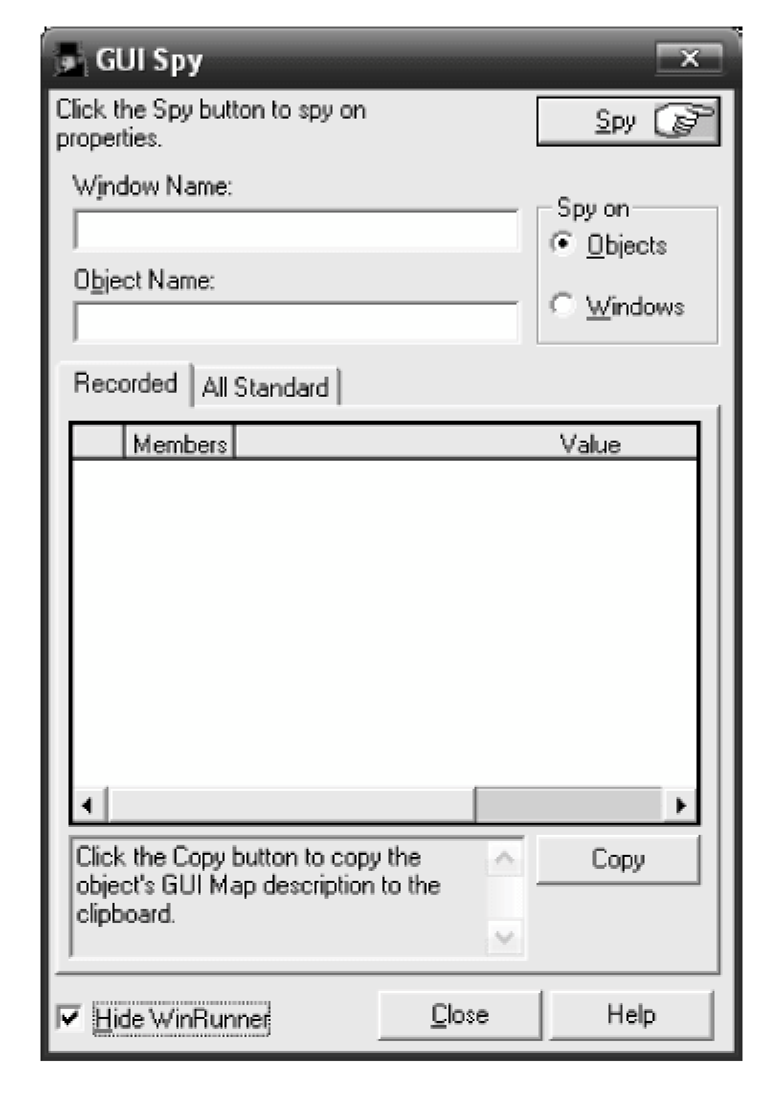


图 6.5 GUI Spy 窗口(1)

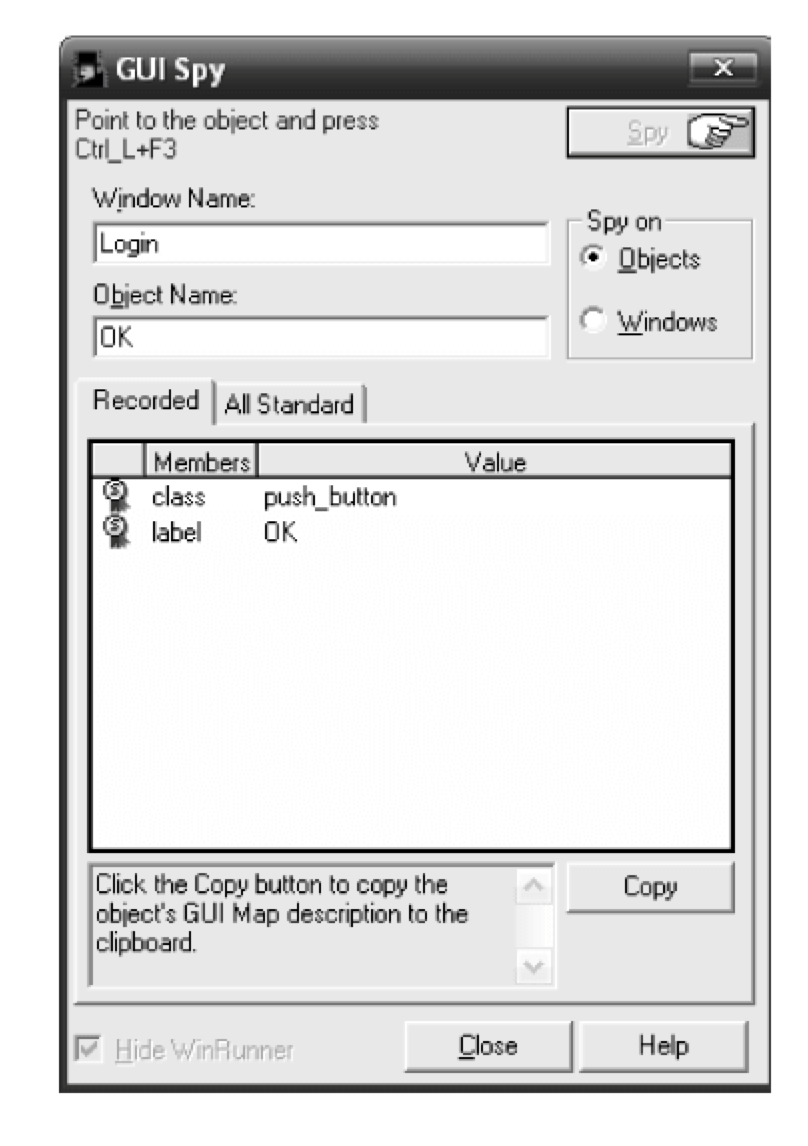


图 6.6 GUI Spy 窗口(2)

5. 检视 GUI Spy 显示的信息

在 GUI Spy 最上面显示了这个【OK】按钮所隶属的窗口是 Login 窗口，且此【OK】按钮的 Logic Name 为 OK。

在【Recorded】页签中，则是显示 WinRunner 用来识别【OK】按钮的属性，分别是 class：push_button 以及 label：OK，表示这个 GUI 对象是个按钮，按钮上面的文字是 OK。

在【All Standard】页签中，则是显示【OK】按钮的所有属性。

在这会发现 WinRunner 只用最少的属性来识别 GUI 对象。

6. 检视 Login 窗口上其他 GUI 对象的属性

花些时间，用 GUI Spy 检视一下 Login 窗口上其他 GUI 对象的属性。

7. 关闭 GUI Spy

单击【Close】按钮关闭 GUI Spy。

6.3.2 GUI Map File 模式

当 WinRunner 识别完 GUI 对象后，会将 GUI 对象储存在 GUI Map File 中，WinRunner 提供两种 GUI Map File 模式：GUI Map File per Test 与 Global GUI Map File。因此在开始使用 WinRunner 识别 GUI 对象并执行自动测试之前，应该先考虑要使用哪种 GUI Map 模式，是 GUI Map File per Test 还是 Global GUI Map File。

在 GUI Map File per Test 模式，当新建立一个测试脚本（test script），WinRunner 就会自动帮助建立此测试脚本的 GUI Map File，当储存测试脚本时，WinRunner 也会自动储存 GUI Map File，而当开启测试脚本时，其 WinRunner 也会自动加载其 GUI Map File，总之所有与 GUI Map File 有关的动作，都由 WinRunner 自动处理。如果是刚开始接触 WinRunner，可以考虑使用 GUI Map File per Test 模式，这种模式就不需要处理 GUI Map File 的相关动作，如建立、储存与加载。

在 Global GUI Map File 模式下，可以多个测试脚本共享一个 GUI Map File。另外，需要储存 GUI Map File，并且在开启测试脚本时，也要同时加载使用的 GUI Map File。如果已经熟悉 WinRunner 的使用，可以考虑使用 Global GUI Map File 模式。

表 6.1 所示为两种模式的比较。

表 6.1 GUI Map File per Test 模式与 Global GUI Map File 模式比较

两种模式	GUI Map File per Test	Global GUI Map File
方法	在测试的过程中将自动保存 GUI 信息，打开测试时可以自动加载 GUI 文件	在测试的过程中需要保存 GUI，当应用程序改变时必须更新 GUI 文件
优点	1. 每个测试都有自己的 GUI 文件 2. 不必保存或加载 GUI 3. 创建简单（录制时自动产生）	1. 当对象或窗体的描述改变，只需把 GUI 文件里对应的属性作相应的修改 2. 容易维护和更新（无须重录）
缺点	只要应用程序的 GUI 改变，每个测试的 GUI 文件都要重录	当新建 GUI 或运行测试脚本时必须保存或装载 GUI 文件
适用范围	适用于初学者或被测软件的 GUI 不会产生变化	适用于经验丰富的 WinRunner 使用者，或被测软件的 GUI 可能会经常产生变化

1. GUI Map File 模式设置

WinRunner 默认值是使用 Global GUI Map File，要设定 GUI Map File 模式，选择【Tools】→【General Options...】→【General】→【GUI Files】命令，设定 GUI Map File 模式，单击【OK】按钮即可，如图 6.7 所示。

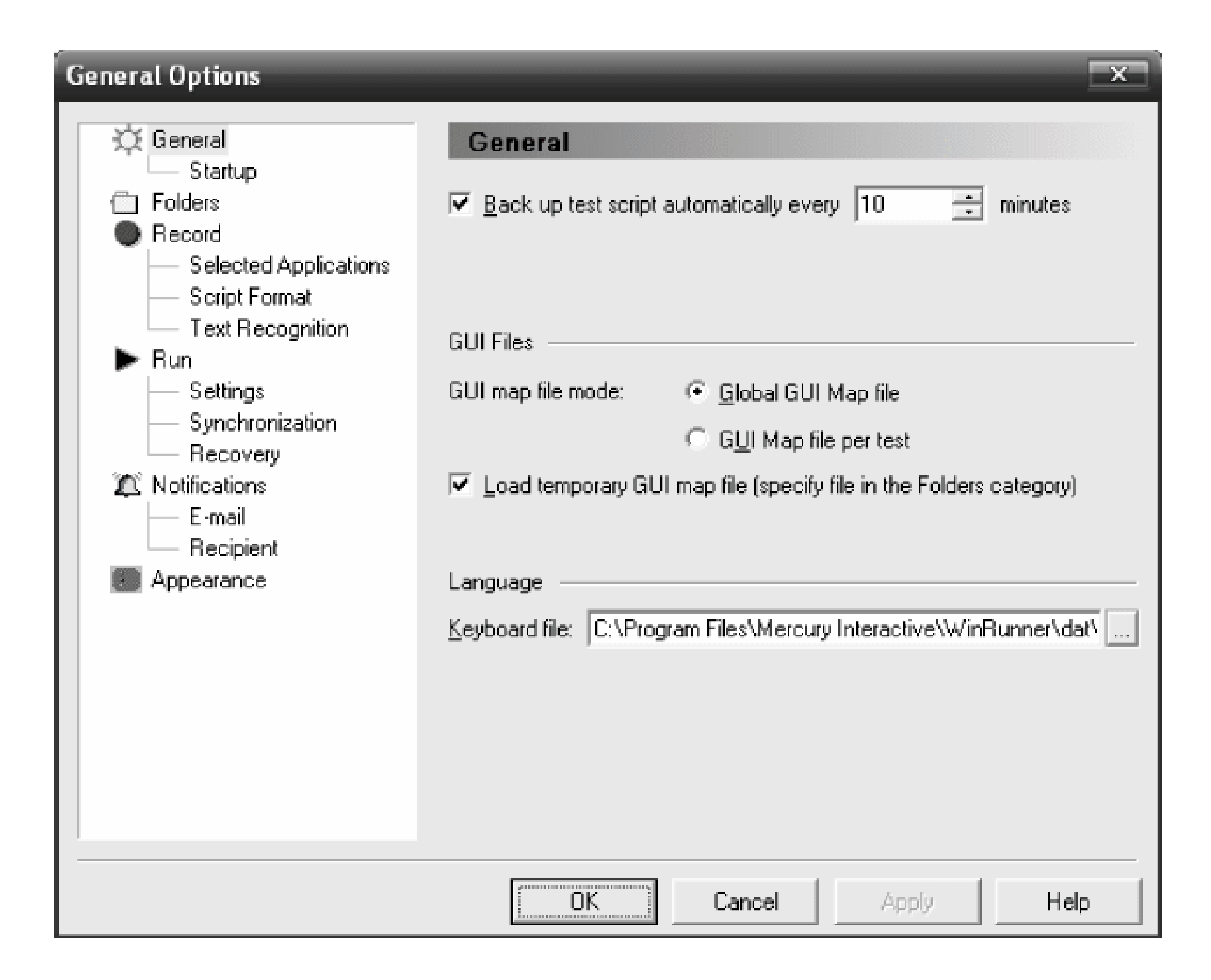


图 6.7　General Options 对话框

注意：如果重新设定 GUI Map File 模式，要重新启动 WinRunner 让设定生效。

2. Global GUI Map File 模式使用

WR 最有效率的用法是把测试分组。一组中的测试(任务)都测试同一窗体上的 GUI 对象。这样这些任务就可以共享 GUI map file。当 GUI 发生变化，只需要修改一个 GUI map file，就可以让同组中的任务都正常工作。

在一个测试组中，某个测试(任务)可能只测试一个窗体内的特定几个 GUI 对象，而另外一个测试(任务)测试这个窗体中的上述对象中的一部分再加上其他对象(也必须在这个窗体中)。因此，如果只使用录制的方法让 WR 学习对象，WR 或许不能把窗体上所有的对象都学到(因为有的对象没有被操作)。最好的方法是在录制前让 WR 全面学习被测软件中所有的 GUI 对象。

WR 有几种学习被测软件 GUI 的方法。通常使用 RapidTest Script wizard 在录制脚本前一次性的学习所有的 GUI 对象。这些 GUI 对象的物理描述保存在 GUI map 文件里。因为这些文件可以共享，其他测试人员就不需要再单独把 GUI 学习一次。如果在软件开发过程中 GUI 发生变化，可以用 GUI Map Editor 来学习单独的窗体或对象，并以此更新 GUI map。测试人员也可以在录制脚本过程中让 WR 自动学习那些被操作的窗体或对象。这种方法比较快且简单，但不是系统性的方式，如果想做全面的测试最好不要用这种方法代替 RapidTest Script wizard。

注意：由于 GUI map File 独立于测试脚本，所以测试人员关闭测试时系统不会自动保存。测试人员一旦对 GUI map 作出修改要记住保存。同样在测试开始时，这些 GUI map File 也不会被自动加载。测试人员可以用 GUI Map Editor 手工加载 GUI map 文件，有效的方法是在脚本开始部分插入 GUI_load 语句，这样脚本就会自动加载相关的 GUI map 文件了。

当测试人员在 Global GUI Map File 模式下工作，如果加载在 GUI Map File per Test 模式下创建的和 GUI 对象相关的测试，这些测试运行起来可能会有问题。

如前文所述,测试人员可以设计让多个测试任务共享同一个 GUI map。举例来说,假设在一个 Open 对话框中 Open button 改成了 OK button。测试人员只要在 GUI map 中修改 Open button 的物理描述,把 button 的 label property 从 Open 改成 OK,修改如下:Open button:{class:push_button,label:OK}。

在测试过程中,当 WR 在测试脚本中遇到 Open 对话框中的逻辑名"Open"时,就搜索含有 label"OK"的 push button。

测试人员可以使用 GUI Map Editor 随时编辑逻辑名和物理描述。可以用 Run wizard 在测试过程中更新 GUI map。如果在运行测试时,WR 找不到某个对象,就自动打开 Run wizard。

WinRunner 学习 GUI 的方法如下。

1) 使用 RapidTest Script Wizard 学习 GUI

在 Global GUI Map File 模式下,可以使用 RapidTest Script Wizard 帮助快速建立 GUIMap File。

在录制脚本前使用 RapidTest Script Wizard 一次性学习被测软件所有的 GUI 对象,生成并保存 GUI map 文件后在脚本开头部分使用 GUI_load 语句加载这个 GUI map 文件。

RapidTest Script Wizard 的使用方法如下。

(1) 选择【Insert】→【RapidTest Script Wizard】命令,单击【Next】按钮,如图 6.8 所示。

注意:当测试人员载入 WebTest 插件或其他某些插件后,RapidTest Script Wizard 将被禁用。

(2) 标识被测软件屏幕打开。

单击 [手形图标] 按钮,然后选择被测软件。被测软件的窗体名称显示在 Window Name 框中,如图 6.9 所示。



图 6.8 RapidTest Script Wizard 的使用方法 1

(3) 选择测试屏幕打开。

(4) 选择测试人员希望 WR 创建的测试类型。当 RapidTest Script Wizard 在被测软件中走查结束,测试人员选择的测试就会显示在 WR 窗口中,如图 6.10 所示。

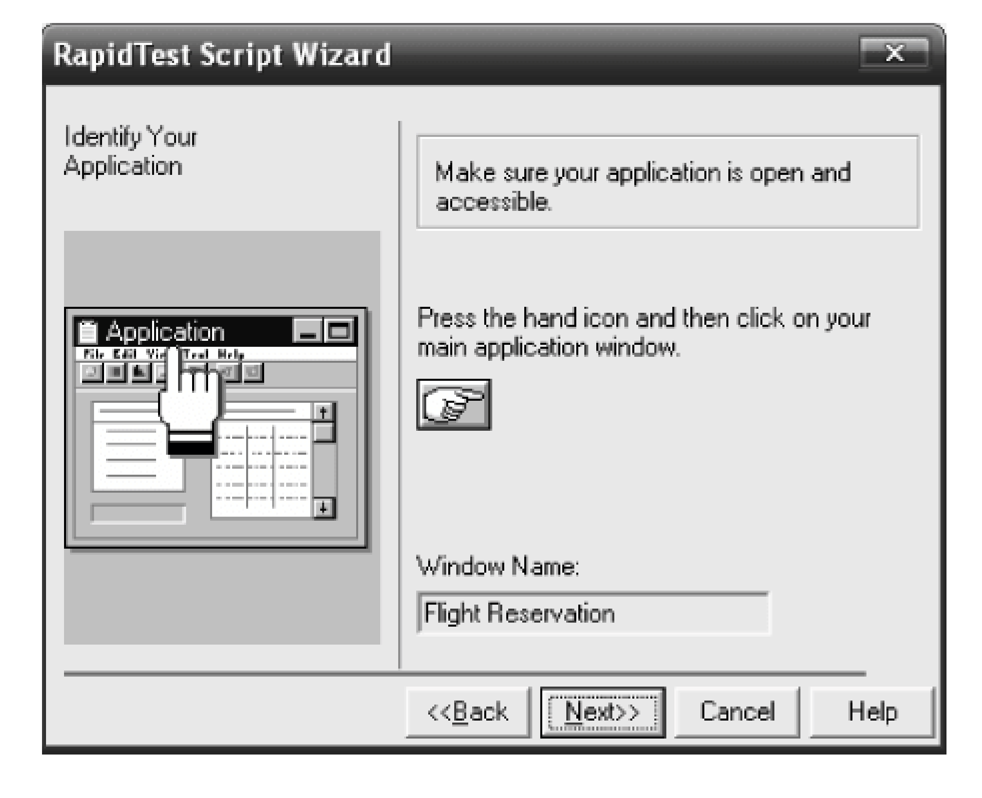


图 6.9 RapidTest Script Wizard 的使用方法 2

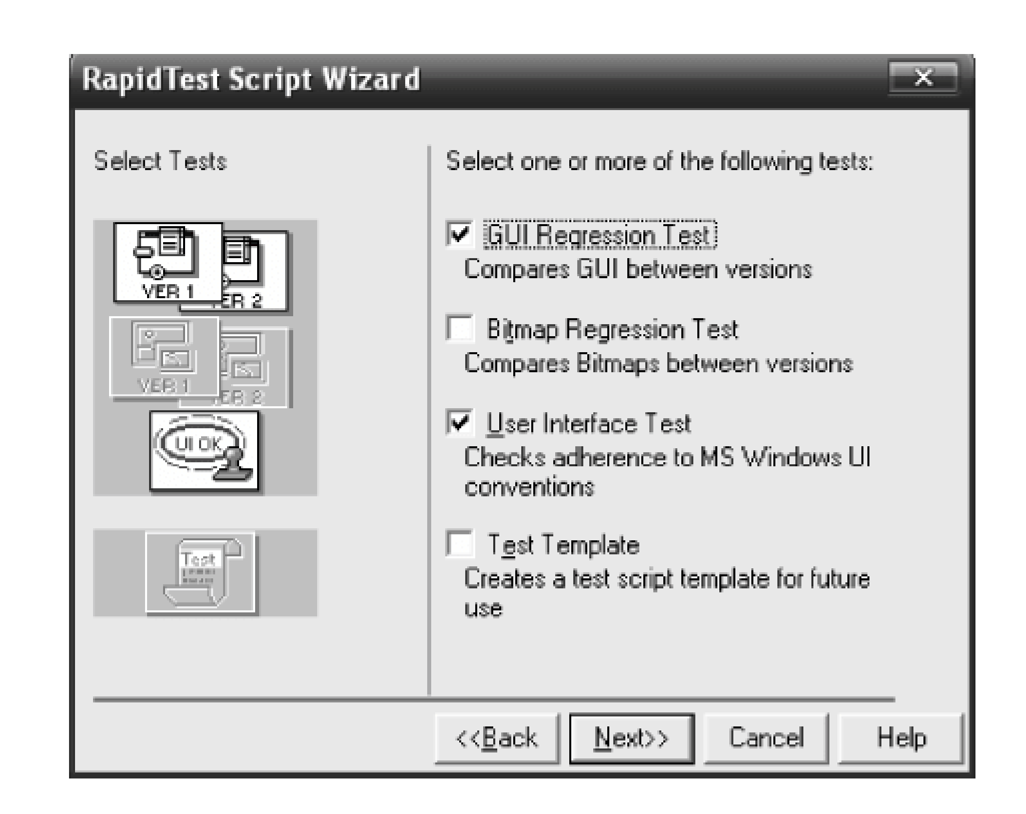


图 6.10 RapidTest Script Wizard 的使用方法 3

测试人员可以选择以下几种类型的测试。

① GUI Regression Test(界面回归测试)：用来比较软件不同版本中的 GUI 对象，例如检查一个 button 是否被禁用。创建这种测试时，WR 先捕捉 GUI 对象默认信息。在回归测试时，WR 把当前信息和默认比较，并报告不符合的地方。

② Bitmap Regression Test(位图回归测试)：用来比较软件不同版本中的位图图片。如果被测软件没有 GUI 对象，则选择这种类型。创建这种测试时，WR 先捕捉被测软件每个窗体的一幅位图图片。在回归测试时，WR 把当前图片和以前捕捉的相比较，并报告不符合的地方。

③ User Interface Test(用户界面测试)：这种测试决定被测软件是否符合 Microsoft Windows 标准。它检查以下几项。

- GUI 对象在窗体中的排列；
- 所有被定义的文本(text)在 GUI 对象上可见；
- GUI 对象上的卷标(Label)以大写字母表示；
- 每个卷标包含一个有下划线的字母；
- 每个窗口有一个 OK button，一个 Cancel button 和一个系统菜单；
- 在这种测试中，WR 搜索软件 GUI，把不符合 Microsoft Windows 标准的地方报告出来。

④ Test Template(测试模板)：这种测试提供一个操作被测软件的自动测试的基本框架。它打开和关闭每个窗口，为测试人员留下可以添加代码(手写或录制)的空间。

注意：即使测试人员不想创建以上任何类型的测试，测试人员仍然可以用 RapidTest Script Wizard 来学习被测软件的 GUI。

(5) 定义导航控制(Navigation Control)。

输入在被测软件中用作导航作用的字符。如果测试人员需要在被测软件的每个窗口暂停以确认用于打开其他窗口的对象，可以选中【Pause to confirm for each window】复选框，如图 6.11 所示，单击【Next】按钮。

(6) 选择【Express】(快速)或【Comprehensive(Advanced)】(全面)学习流程，如图 6.12 所示。单击【Learn】按钮，WR 就开始系统地一个窗口一个窗口地学习被测软件。这个过程的时间长短取决于被测软件的复杂程度。

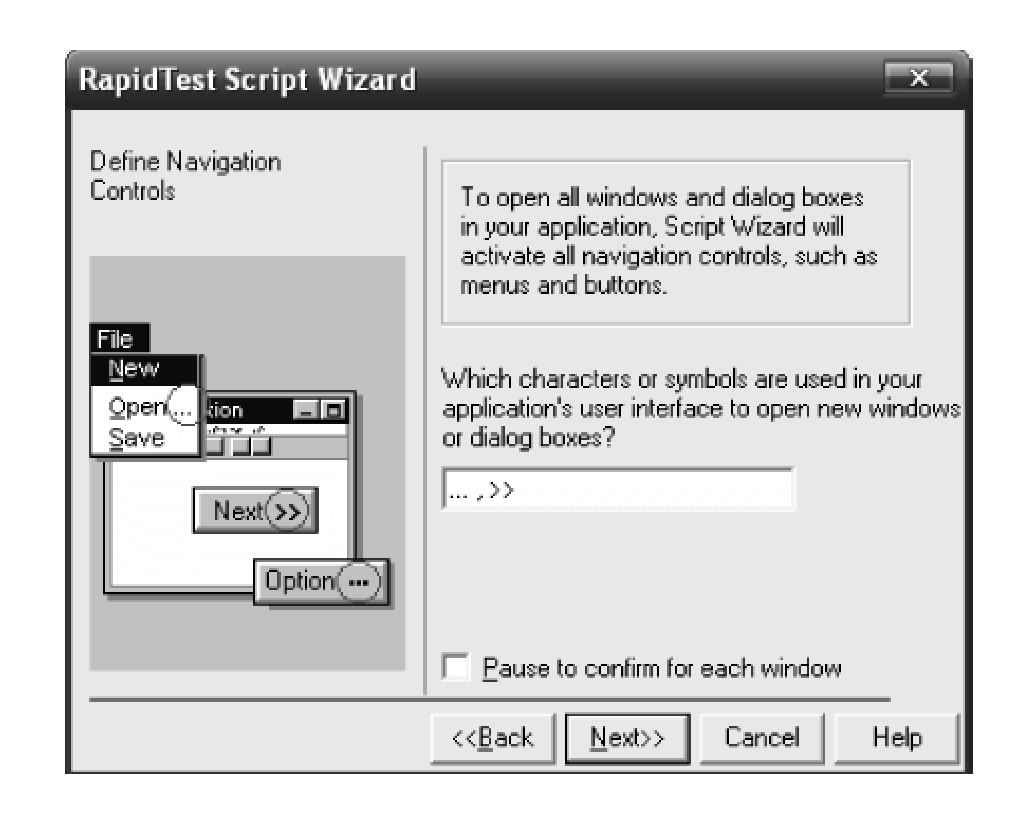


图 6.11　RapidTest Script Wizard 的使用方法 4

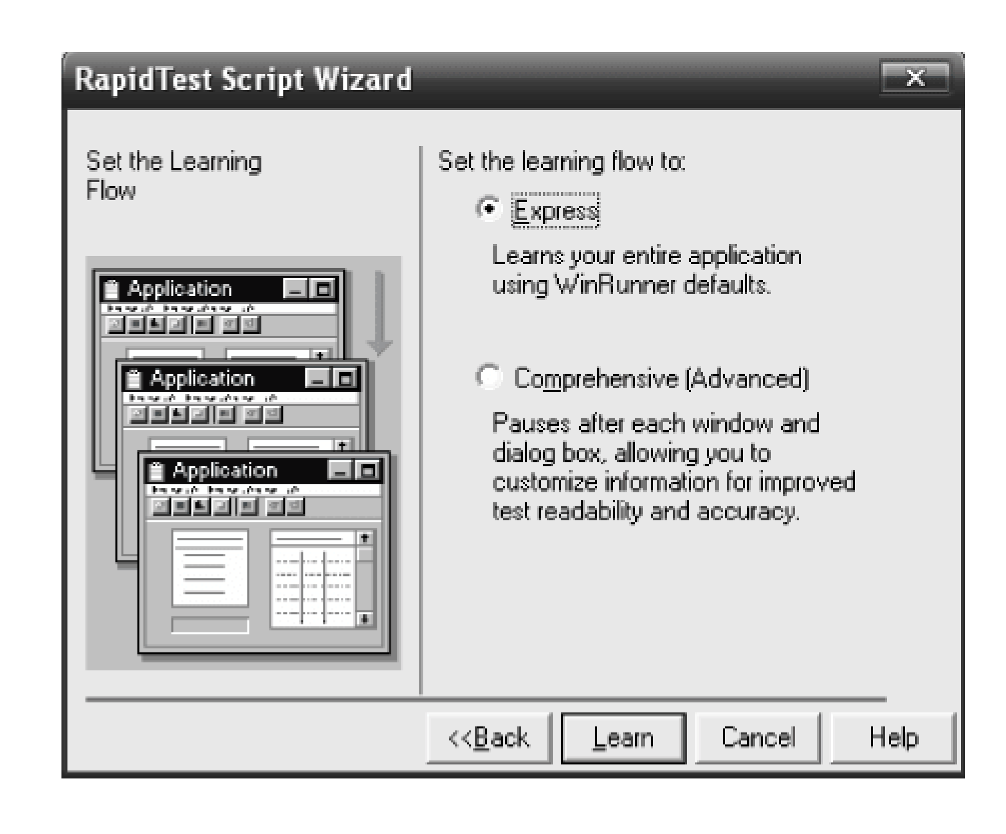


图 6.12　RapidTest Script Wizard 的使用方法 5

(7) 选择【Yes】或【No】按钮来告诉 WR 测试人员是否希望在使用 WR 时,让 WR 自动启动这个被测软件,然后单击【Next】按钮。

(8) 输入启动脚本和 GUI map 文件的保存路径和文件名,或使用默认值,然后单击【Next】按钮。

(9) 输入测试文件的保存路径和文件名,或使用默认值,然后单击【Next】按钮。

(10) 单击【OK】按钮关闭 RapidTest Script Wizard。测试人员刚才创建的测试被显示在 WR 窗口中。

2) 使用录制方式学习 GUI

WR 也可以通过在 Context Sensitive 模式(默认模式)下录制脚本的方法学习 GUI 对象。测试人员只需要录制对被测软件的操作,WR 会自动学习操作中碰到的 GUI 对象。

当测试人员开始录制时,WR 先检查对象是否已经存在于 GUI map 中,如果没有,就学习这个对象。

WR 先把学到的信息放在临时 GUI map 文件中,因此测试人员在关闭 WR 时要记住保存。

如果测试人员不希望 WR 把学到的信息添加到临时 GUI map 文件中,测试人员可以在 General Options 对话框的 General 栏设置让 WR 不加载临时 GUI map 文件。总的来说,录制方式只用于小的或临时的测试。

3) 使用 GUI Map Editor 学习 GUI

(1) 选择【Tools】→【GUI Map Editor】命令打开 GUI map 编辑器。

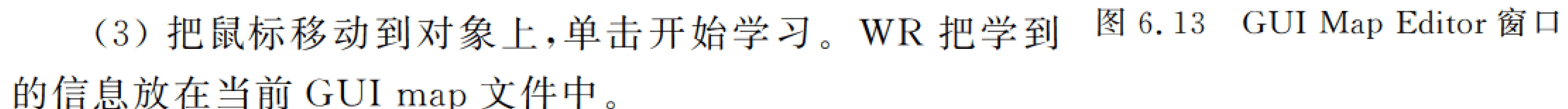
(2) 单击【Learn】按钮,如图 6.13 所示。

想要学习一个窗体中所有的对象,就单击窗体的标题栏。当提示是否学习窗体中所有对象时,单击【是】按钮。

如果只想学习窗体,就点击窗体的标题栏。当提示是否学习窗体中所有对象时,单击【否】按钮。

如果只学习个别对象,就单击这个对象。(取消操作,右击)。

(3) 把鼠标移动到对象上,单击开始学习。WR 把学到的信息放在当前 GUI map 文件中。



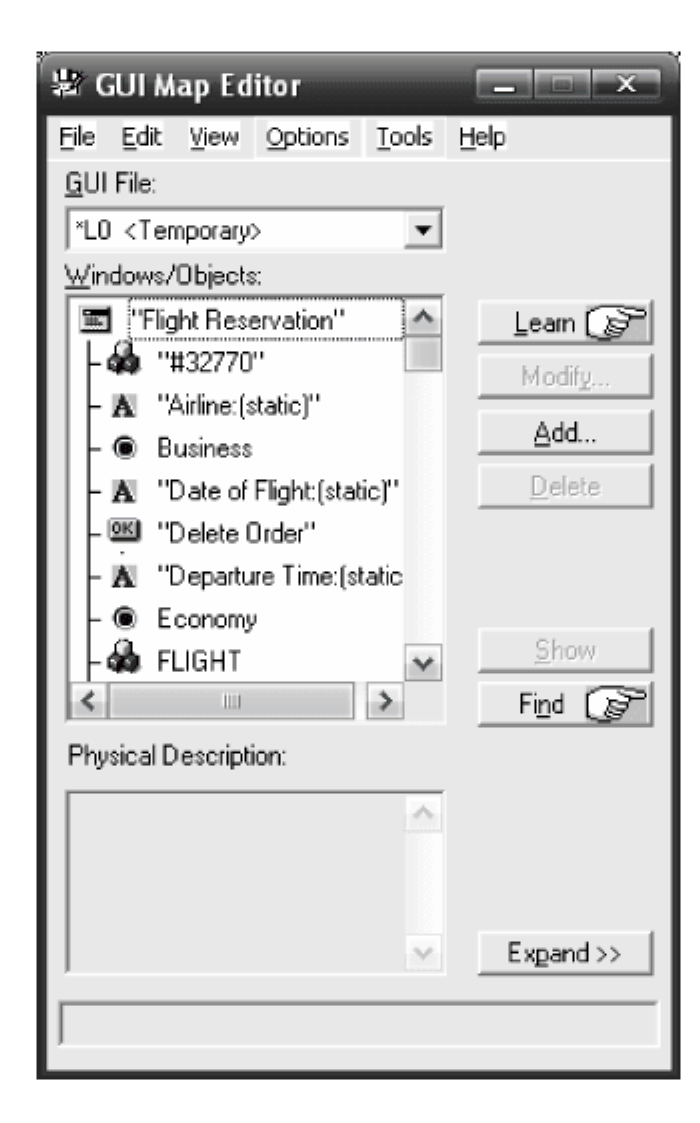


图 6.13 GUI Map Editor 窗口

3. GUI Map File per Test 模式的使用

使用 GUI Map File per Test 模式,测试人员不需要 WR 去学习被测软件的 GUI,也不需要保存或加载 GUI map 文件,WR 会自动完成上述的一切。WR 在测试人员创建新测试的时候自动创建一个新的 GUI map 文件;在测试人员保存测试的时候自动保存 GUI map 文件;在测试人员打开测试时自动加载 GUI map 文件。

在【General Options】对话框中的 General 栏可以选择使用【GUI Map file per test】模式。WR 重新启动后设置才会生效,如图 6.14 所示。

在这种模式下,WR 通过录制的方式学习被测软件的 GUI。如果 GUI 发生变化,测试人员可以用 GUI Map Editor 更新每个测试的 GUI map。测试人员无需加载或保存 GUI map 文件。

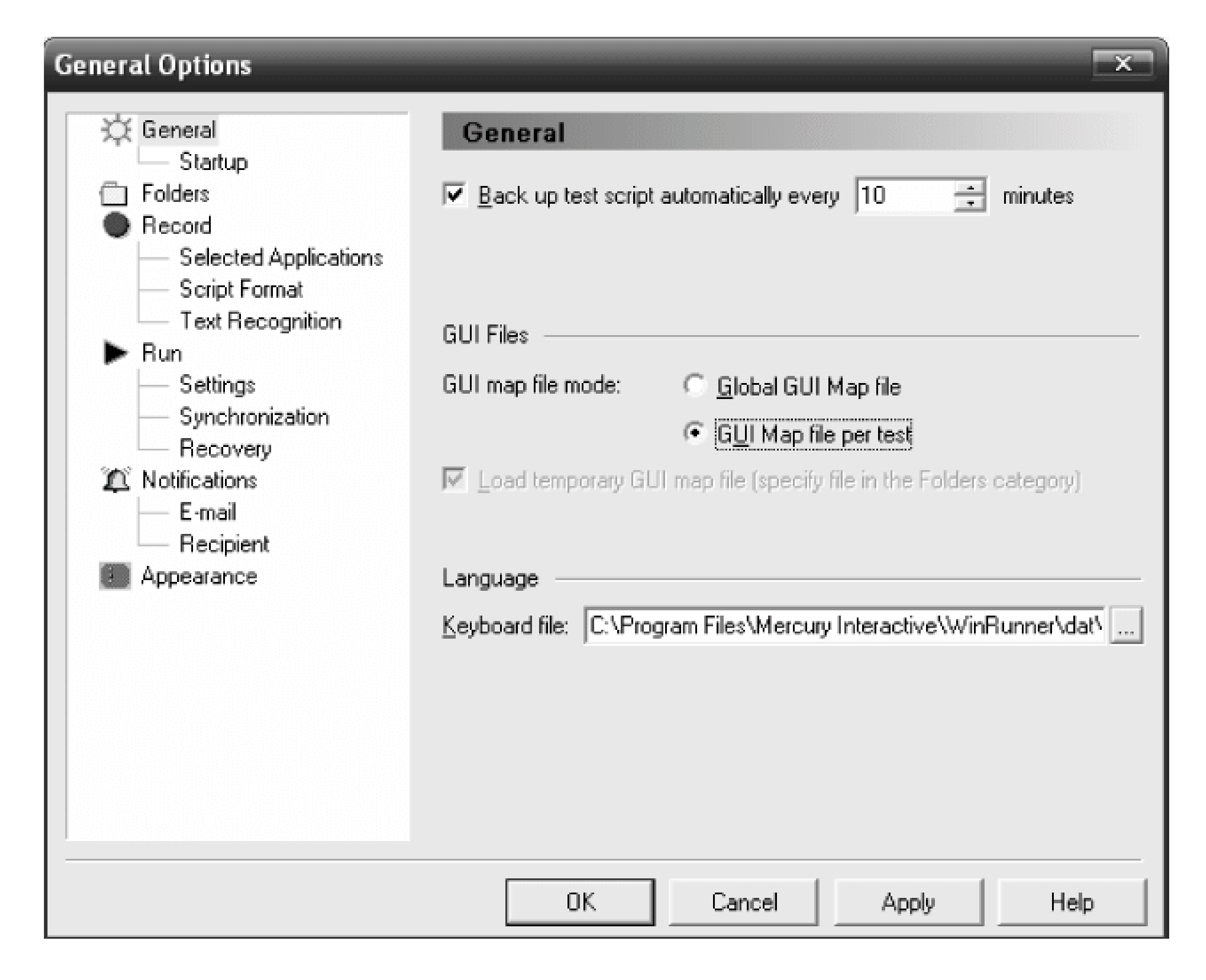


图 6.14 【General Options】对话框

注意：如果改变了对象的逻辑名，必须更新脚本。不要在 GUI Map Editor 里保存对 GUI map 文件的修改，测试人员保存测试时，这些变更会被自动保存。不要手工加载或卸除 GUI map 文件。这些文件在测试人员打开测试时会被自动加载。

6.4 录制测试脚本

6.4.1 选择录制模式

表 6.2 列出的内容可以帮助测试人员决定使用哪种录制模式。

表 6.2 Context Sensitive 和 Analog 模式比较

Context Sensitive	Analog
识别应用程序的 GUI 对象(用户界面)	应用程序包含位图的范围(和画图的范围)
不能记录精确的鼠标的动作	能够精确的记录鼠标的动作

6.4.2 Context Sensitive 模式下录制

测试人员以 Context Sensitive 模式录制一段测试脚本，此测试脚本的操作流程为在 Flight Reservation 开启一笔订单。

1. 开启 WinRunner 并加载 GUI Map File

执行【开始】→【程序】→【WinRunner】→【WinRunner】命令，如果是第一次执行 WinRunner，会开启欢迎窗口，则选择【New Test】项；如果没有开启欢迎窗口，则选择【File】→【New】命令。

检查 GUI Map File 是否已经加载，选择【Tools】→【GUI Map Editor】命令开启 GUI Map Editor，再选择【View】→【GUI Files】命令检查是否加载 flight4a.gui 文件。如果 flight4a.gui 没有加载，选择【File】→【Open】命令然后选取 flight4a.gui 文件，单击【Open】

按钮将其载入。

2. 开启 Flight Reservation 并登录

执行【开始】→【程序】→【WinRunner】→【Sample Applications】→【Flight 4A】命令，登录窗口会开启。在【Agent Name】文本框中输入名字(至少四个英文字母)，在【Password】文本框中输入“Mercury”，单击【OK】按钮登录到 Flight Reservation。

调整 WinRunner 与 Flight Reservation 的窗口大小与位置，使这两个窗口不重叠。

3. 开始以 Context Sensitive 模式录制测试脚本

在 WinRunner 中选择【Test】→【Record-Context Sensitive】命令或单击工具栏上的 Record 按钮，从现在开始 WinRunner 会录制所有鼠标的点选以及键盘的输入。请注意 Record 会变成 Record，表示现在已经进入 Context Sensitive 录制模式了。在 WinRunner 下方的状态栏同样也会有变化，表示现在已经在录制测试脚本了。

4. 开启 2 号订单

在 Flight Reservation 中选择【File】→【Open Order】命令，在【Open Order】对话框中选中【Order No.】复选框并且输入“2”，单击【OK】按钮，如图 6.15 所示。



图 6.15　Open Order 窗口

5. 停止录制

在 WinRunner 中选择【Test】→【Stop Recording】命令，或单击工具栏上的 Stop 按钮，停止录制测试脚本。

6. 储存测试脚本

选择【File】→【Save】命令或单击工具栏上的 按钮，将测试脚本储存成 ex2。

在之前的练习中，录制了在 Flight Reservation 中开启订单的测试脚本，WinRunner 产生了以下的测试脚本。

```
# Flight Reservation
    set_window("Flight Reservation",3);
    menu_select_item("File; Open Order...");

# Open Order
    set_window("Open Order",1);
    button_set("Order No.",ON);
    edit_set("Edit","2");
    button_press("OK");
```

以下是对此段测试脚本进行的详细说明。

(1) 当测试人员选择一个 GUI 对象，WinRunner 会自动为这个 GUI 对象取个名字，通常是以 GUI 对象上的文字作为名字，此名字在 WinRunner 中称为 logic name。这个 logic name 可以让测试人员更容易的阅读测试脚本。例如当测试人员选择【Order No.】复选框时，WinRunner 产生以下的指令：

```
button_set("Order No.",ON);
```

Order No. 就是这个【Order No.】check box 的 logic name。

(2) 当测试人员换到另一个窗口上操作时，WinRunner 会自动在测试脚本上加上一行批注，帮助测试人员更容易阅读测试脚本。例如，当点选 Flight Reservation 窗口时，WinRunner 会自动加上下面的注解：

```
#Flight  Reservation
```

(3) 当测试人员换到另一个窗口上操作时，WinRunner 会自动产生一行 set_window 指令，然后才是它操作的指令。例如当测试人员开启 Open Order 窗口时，WinRunner 会先产生下面的指令：

```
set_window("Open Order",1);
```

(4) 当测试人员以键盘输入时，WinRunner 会产生 type、obj_type 或是 edit_type 等指令。例如当测试人员在 Order No. 文本框中输入"2"时，WinRunner 会产生下面的指令：

```
edit_set("Edit","2");
```

6.4.3 Analog 模式下录制

测试人员以 Analog 模式录制一段测试脚本，此测试脚本的操作流程为在 Flight Reservation 下传真一笔订单。开始会以 Context Sensitive 的模式录制，然后在签名的时候切换为 Analog 的模式录制测试脚本，录制完签名的部分，再切换回 Context Sensitive 的模式。

(1) 开启 ex2 测试脚本，并将光标移到最后一行。接下来的操作将以 ex2 测试脚本继续录制下去。选择【File】→【Open】命令开启 ex2 测试脚本，并且将光标移到最后一行。

(2) 开始以 Context Sensitive 模式录制测试脚本。在 WinRunner 工具中选择【Test】→【Record-Context Sensitive】或单击工具栏上的 Record 按钮。

(3) 开启传真订单。在 Flight Reservation 中选择【File】→【Fax Order】命令，在【Fax Number】文本框中输入"0288303456"，如图 6.16 所示。

(4) 选中【Send Signature with order】复选框。

(5) 在 Context Sensitive 模式下录制签名动作。用鼠标在【Agent Signature】空白区域中签名，这时请注意 WinRunner 如何录制签名动作。

(6) 清除签名。单击【Clear Signature】按钮。

(7) 将 Fax Order 窗口移动到其他位置。在切换到 Analog 模式之前，移动一下 Fax Order 窗口。



图 6.16 Fax Order No. 2 窗口

(8) 在 Analog 模式下录制签名动作。按键盘上的 F2 键或单击工具栏上的 Record 按钮，此时录制模式将从 Context Sensitive 切换到 Analog 模式，且 Record 会变成 Record，表示现在是 Analog 模式。用鼠标在【Agent Signature】空白区域中签名，这时请注意 WinRunner 如何录制签名动作。

(9) 切换回 Context Sensitive 模式并将订单传真出去。按键盘上的 F2 键或单击工具

栏上的 Record 按钮，此时录制模式会从 Analog 模式切换回 Context Sensitive 模式。单击【Send】按钮后，Flight Reservation 会仿真将订单传真出去。

(10) 停止录制。在 WinRunner 中选择【Test】→【Stop Recording】命令，或单击工具栏上的 Stop 按钮停止录制测试脚本。

(11) 储存测试脚本。选择【File】→【Save】命令或单击工具栏上的按钮。

(12) 在 Global GUI Map File 模式下，要储存新的 GUI 对象。在前面已经以 RapidTest Script Wizard 识别过 Flight Reservation 的 GUI 对象，但是因为 Fax Order 这个窗口必须在开启一笔订单后才能再开启 Fax Order 窗口，因此 RapidTest Script Wizard 并未将 Fax Order 窗口的 GUI 对象也识别下来。所以当录制到 Fax Order 窗口上的操作时，WinRunner 会识别到新的窗口与 GUI 对象，并且先放到 temporary GUI Map File 中。但是当关闭 WinRunner 后，在 temporary GUI Map File 中的 GUI 对象将会被抛弃，所以一定要将新识别的 GUI 对象储存下来，这个动作非常重要，一定要记得。选择【Tools】→【GUI Map Editor】命令，再选【View】→【GUI Files】命令，测试人员可以看到 Fax Order No. 2 窗口是放在 L0＜temporary＞GUI Map File 中。选择【File】→【Save】命令，选取 flight4a. gui 文件，单击【OK】按钮，则 Fax Order No. 2 窗口以及其 GUI 对象，将会被储存到 flight4a. gui 文件中。最后关闭 GUI Map Editor。

6.4.4 测试脚本执行

完成上面的练习之后，已经准备好执行测试脚本并分析测试结果了。WinRunner 提供 3 种执行测试脚本的模式：Verify、Debug、Update。

Verify：当测试人员真正执行测试以检查应用程序的功能，并且要储存测试结果时使用。

Debug：当测试人员想检查测试脚本执行是否流畅，没有错误时使用。

Update：当测试人员要更新检查点的预期值时使用。

测试脚本执行过程如下。

(1) 确认 WinRunner 与 Flight Reservation 的主窗口都已经开启。

(2) 开启测试脚本。选择【File】→【Open】命令开启 ex2 测试脚本。

(3) 检查 Flight Reservation 是否在主窗口中。如果有其他对话窗口请先关闭。

(4) 确认工具栏上显示 Verify 模式。

(5) 选择【Run From Top】命令。选择【Test】→【Run From Top】命令或单击工具栏上的 From Top 按钮，则 Run Test 窗口将会开启，单击【OK】按钮开始执行测试。

(6) 输入 Test Run Name。输入 Test Run Name，WinRunner 会将测试脚本执行的结果储存在 Test Run Name 的目录下，如 res1，而此测试结果将会储存在测试脚本目录下。

如选中窗口下方【Display test results at end of run】复选框，则当测试脚本执行完毕后，WinRunner 会自动开启测试执行结果的窗口，如图 6.17 所示。

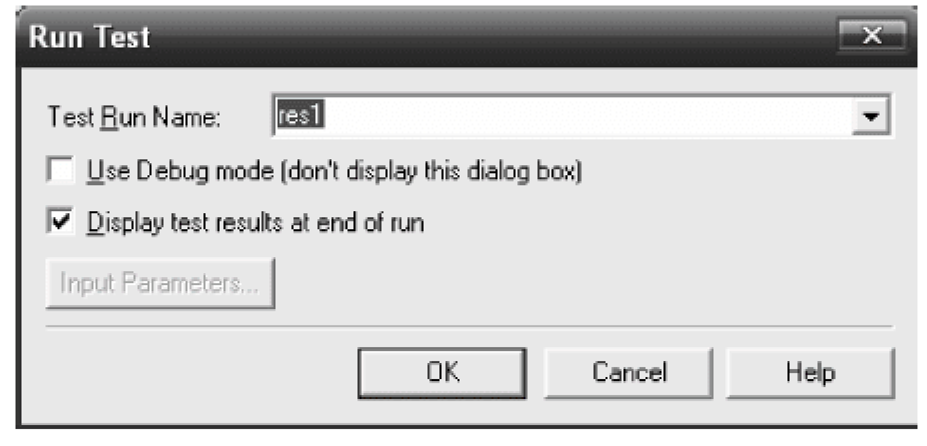


图 6.17　Run Test 窗口

(7) 执行。单击【OK】按钮后 WinRunner 会

开始执行测试脚本。请注意观察 WinRunner 如何执行测试脚本。

(8) 检视执行结果。当测试执行完毕后，WinRunner 会开启 Test Results 窗口，显示测试执行的结果。

6.4.5 测试结果分析

当执行完测试脚本，就可以检视测试结果。

WinRunner 提供以下两种类型的测试结果检视器。

(1) WinRunner Repor：一般 GUI 接口的检视器。

(2) Unified Repor：HTML 类型的检视器。

6.4.6 录制时建议

(1) 录制前请先关闭不必要的应用程序或窗口。

(2) 尽量在录制结束时，回到开始录制的画面，以便测试脚本可以重复执行测试。例如当测试人员从主窗口开始录制测试脚本时，在测试脚本的最后，还是要回到主窗口画面。

(3) 当以 Analog 模式录制时，尽量避免录制按住鼠标的动作。例如当要卷动窗口画面时，以 click 的方式卷动窗口，尽量不要以按住 scroll bar 拖曳的方式卷动窗口。

(4) 当需要从 Context Sensitive 模式切换到 Analog 模式时，在切换前建议移动一下窗口，如此可确保以 Analog 模式录制完成后执行时，窗口位置为固定的。

(5) 当录制点选非标准的 GUI 对象时，WinRunner 会产生 obj_mouse_click 的指令。例如当测试人员点选一个图像对象时，WinRunner 可能会产生下列的指令：

```
obj_mouse_click(GS_Drawing,8,53,LEFT);
```

假如这个非标准 GUI 对象的行为与标准的 GUI 对象类似，如其功能与按钮一样，则可以通过对应(map)的方式，将其对应到标准的 GUI 对象，如此一来 WinRunner 便会以标准 GUI 对象的指令，如 button_press 来取代 obj_mouse_click 的指令了。

(6) 当在 Global GUI Map File 模式下录制测试脚本时，录制的 GUI 对象之前并未录制过，则 WinRunner 会将其放在 temporary GUI Map File 中。

(7) 在录制过程中可以利用 F2 键切换 Context Sensitive 与 Analog 的录制模式。

(8) 当在 Global GUI Map File 模式下录制测试脚本时，记得经常检查新的 GUI 对象是否被新增到 temporary GUI Map File 中。当关闭 WinRunner 之前请记得将存放在 temporary GUI Map File 中的 GUI 对象存盘。

6.5 同 步 点

当测试人员执行测试时，所测试的应用程序每次操作的响应时间并不一定，有时快，有时慢，导致执行输入动作的时间也需要等待。例如，以下的动作常会花几秒钟。

- 从数据库取得数据
- 等待一个窗口开启
- 等待状态列成为 100%

• 等待某个状态信息出现

当遇到这类的情况，WinRunner 会等待一段固定的时间，直到应用程序接受输入的动作，这个等待时间的默认值为 10 秒钟。假如应用程序响应的时间超过 WinRunner 等待的时间，则测试执行就可能会失败。

如果在测试执行过程中遇到这样的情况，可以用下列的方式解决。

(1) 增加 WinRunner 预设等待的时间选择【Tools】→【General Options...】→【Run】→【Settings】命令，将【Timeout for checkpoints and CS statements】的值加大，预设为“10000msec”。加大这个设定可能会造成在 Context Sensitive 的动作变慢。

(2) 在测试脚本中插入同步点(synchronization point)，当 WinRunner 执行到同步点时，会暂停执行以等待应用程序某些状态的改变后，再继续执行。这个方式也是经常被使用的方式。

实例 2：

(1) 在 Flight Reservation 中建立一张新的订单，并新增到数据库中。

(2) 变更预设等待时间的设定。

(3) 如何识别何种问题需要以同步点解决。

(4) 加入同步点。

(5) 执行测试脚本并检视结果。

1. 录制测试脚本

(前面已经介绍过，此处不做详细讲解)

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 开启 Flight Reservation 并登录。

(3) 开始以 Context Sensitive 模式录制测试脚本。

(4) 建立新的订单。在 Flight Reservation 中选择【File】→【New Order】命令。

(5) 填入航班与旅客资料。请输入以下数据，如图 6.18 所示。

【Date of Flight】：08/01/06(日期格式为 MM/DD/YY，日期要大于今天的日期)

【Fly From】：Denver

【Fly To】：Frankfurt

单击【Flights...】按钮，选取一个航班。

【Name】：swpu

【Class】：Economy

(6) 选择【Insert Order】项，当完成新增订单后，状态列会显示【Insert Done...】的信息。

(7) 选择【Delete Order】命令删除刚刚新增的订单，并单击【是】按钮确认。

(8) 停止录制测试脚本。

(9) 储存测试脚本，将测试脚本储存为 ex3。

2. 变更预设等待时间的设定

WinRunner 预设等待时间为 10 秒钟。为了模拟出需要加入同步点的状况，下面将变更 WinRunner 预设等待时间的设定缩短为 1 秒钟。

(1) 开启【General Options】对话窗口，选择【Tools】→【General Options...】命令。

(2) 选择【Run】→【Settings】命令。



图 6.18 Flight Reservation 窗口

(3) 在【Timeout for checkpoints and CS statements】中将 10 000 改成 1000。

(4) 单击【OK】按钮,关闭对话窗口。

3. 如何识别需要以同步点解决的问题

当执行 ex3 测试脚本时,将会出现同步点的问题。

(1) 执行 WinRunner 并开启 ex3。

(2) 选择【Test】→【Run From Top】命令或单击工具栏上的 From Top 按钮,则 Run Test 窗口将会开启,单击【OK】按钮开始执行测试。

(3) 在测试脚本执行的过程中,请特别注意当 WinRunner 选择【Delete Order】按钮时发生什么事。

(4) 暂停执行。

当 WinRunner 执行到选择【Delete Order】按钮时,由于 Insert Order 的动作尚未完成,而 WinRunner 最多只等待 1 秒钟,所以当 1 秒钟过去后,而【Delete Order】按钮还是 disabled 的状态,造成 WinRunner 无法选择【Delete Order】按钮,并跳出【Object is currently disabled】的对话窗口,表示 WinRunner 要操作的 GUI 对象是 disabled 的,所以无法执行,如图 6.19 所示。



图 6.19 WinRunner 报错窗口

(5) 单击【Pause】按钮。

这时可以发现黄色小箭头停在点选的"Delete Order"这行指令上。

接下来要在 ex3 测试脚本中插入同步点,这个同步点会获取状态列上【Insert Done...】的图像,然后当再次执行测试脚本时,WinRunner 会等到【Insert Done...】的图像出现后,才执行单击【Delete Order】的动作。

(1) 确认 Flight Reservation 已经开启。

(2) 确认 WinRunner 已经开启,并加载 ex3 测试脚本与 GUI Map File。

(3) 将光标移动到要插入同步点的位置。

在 button_press(“Delete Order”)这一行上面插入一行空白行,并将光标移到这一行空白行的开头。

(4) 插入同步点。

选择【Insert】→【Synchronization Point】→【For Object/Window Bitmap】命令,将鼠标光标移动到【Insert Done...】的状态列上并点选,WinRunner 会在测试脚本中插入一行“obj_wait_bitmap("Insert Done...","Img1",1); ”的指令,这一行指令表示当 WinRunner 执行到这里时,会等待【Insert Done...】的图像出现,等待时间为 1 秒钟,当图像出现后,才会继续往下执行。

(5) 手动将 1 秒钟改成 10 秒钟。

由于等待 1 秒钟还是太短,所以手动将“obj_wait_bitmap("Insert Done...","Img1",1); ”指令改成“obj_wait_bitmap("Insert Done...","Img1",10); ”指令等待 10 秒钟。

(6) 储存测试脚本。

(7) 如果在 Global GUI Map File 模式下,记得储存新的 GUI 对象。

由于【Insert Done...】的图像为 WinRunner 新识别的 GUI 对象,所以要记得储存。选择【Tools】→【GUI Map Editor】命令,再选择【View】→【GUI Files】命令,可以看到新识别的 GUI 对象是放在“L0＜temporary＞GUI Map File”中。选择【File】→【Save】命令,选取 flight4a. gui,单击【OK】按钮,则新识别的 GUI 对象,将会被储存到 flight4a. gui 中。最后关闭 GUI Map Editor。

执行已加入同步点的测试脚本,并检视执行结果。

(1) 确认 WinRunner 与 Flight Reservation 的主窗口都已经开启。

(2) 开启 ex3 测试脚本。

(3) 确认工具栏上显示 Verify 模式。

(4) 选择【Test】→【Run From Top】命令或单击工具栏上的 From Top 按钮,则 Run Test 窗口将会开启,接受预设 res2 的执行名称,确认选中【Display test results at the end of run】复选框,单击【OK】按钮开始执行测试。

(5) 检视测试结果。当执行结束,WinRunner 会自动开启测试执行结果。在测试结果下方的事件中,有一行绿色的 wait for bitmap 事件,表示同步点执行成功。可以双击此事件,检视此同步点的图像结果。

(6) 关闭测试结果窗口:选择【File】→【Exit】命令。

(7) 关闭 ex3 测试脚本:选择【File】→【Close】命令。

(8) 关闭 Flight Reservation:选择【File】→【Exit】命令。

(9) 将 WinRunner 预设等待时间改回 10 秒钟。

6.6 GUI 对象检查点

设定检查点可以检查所设定区域的显示是否和预期结果相符。通过检查点的设置以及对各点处输出信息的编程定义,可以在脚本运行结果单中查看各项测试内容是否都已通过。在功能测试中,检查点可以用在以下两个方面:检查应用程序经过修改后对象状态是否发生变化;检查对象数据是否和预期数据一致。WinRunner 提供的检查点类型有 GUI

checkpoint、Bitmap checkpoint、text checkpoint 等。

1. 如何检查 GUI 对象

在测试应用程序时，通常是通过检查 GUI 对象的属性，来测试功能是否正常，当 GUI 对象的属性值与预期的值不符合时，也就表示可能有问题产生。

在 WinRunner 中可以建立 GUI 检查点(checkpoint)，检查 GUI 对象的属性，例如测试人员可以检查以下几项。

- swpu：输入字段的内容。
- Economy：是否被选取。
- Insert Order：按钮是否可用。

要建立单一 GUI 对象的检查点，首先指定要建立检查点的 GUI 对象。

测试人员单击时，WinRunner 会以预设检查的属性建立检查清单(checklist)，并将检查点插入到测试脚本中。

检查清单的内容记录了要 WinRunner 检查的 GUI 对象与属性。

测试人员双击检查清单的内容，【Check GUI】对话窗口会开启并显示测试人员选取的 GUI 对象，以及此 GUI 对象可供检查的属性，测试人员只要在【Check GUI】对话窗口上选中想检查的属性，WinRunner 就会建立检查清单(checklist)，并将检查点插入到测试脚本中。

不管建立的检查点是检查预设的属性还是测试人员选取的属性，WinRunner 会获取建立检查点当时的属性值当作预期的值，并且在测试脚本中插入 obj_check_gui 或 win_check_gui。

当测试人员执行测试脚本时，WinRunner 会自动比较执行时的实际值与建立检查点时的预期值，如果一致，表示检查点检查通过；如果不一致，表示检查点检查失败。

2. 建立 GUI 对象检查点

实例 3：

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 开启 Flight Reservation 并登录。

(3) 开始以 Context Sensitive 模式录制测试脚本。

(4) 开启 Flight Reservation 的【Open Order】窗口。

(5) 对【Order No.】复选框建立检查点。在 WinRunner 中选择【Insert】→【GUI Checkpoint】→【For Object/Window】命令，或选择使用者工具栏上的 按钮。双击【Order No.】复选框，则【Check GUI】对话窗口会开启并显示选取的 GUI 对象，以及此 GUI 对象可供检查的属性。请注意如果单击，则【Check GUI】对话窗口不会开启，且 WinRunner 会直接以【State】属性当成检查点要检查的属性，并插入检查点，如图 6.20 所示。

单击【OK】按钮，WinRunner 会在测试脚本中插入 obj_check_gui 检查点。

(6) 输入订单编号 3。在【Open Order】窗口中，选中【Order No.】复选框，并且在字段中输入“3”。

(7) 对【Order No.】复选框建立另一个检查点。在 WinRunner 中选择【Insert】→【GUI Checkpoint】→【For Object/Window】命令，或单击使用者工具栏上的 按钮。单击【Order No.】复选框，WinRunner 会马上以预设的属性(status)在测试脚本中加上检查点(obj_check_gui)，其预期值为 ON。

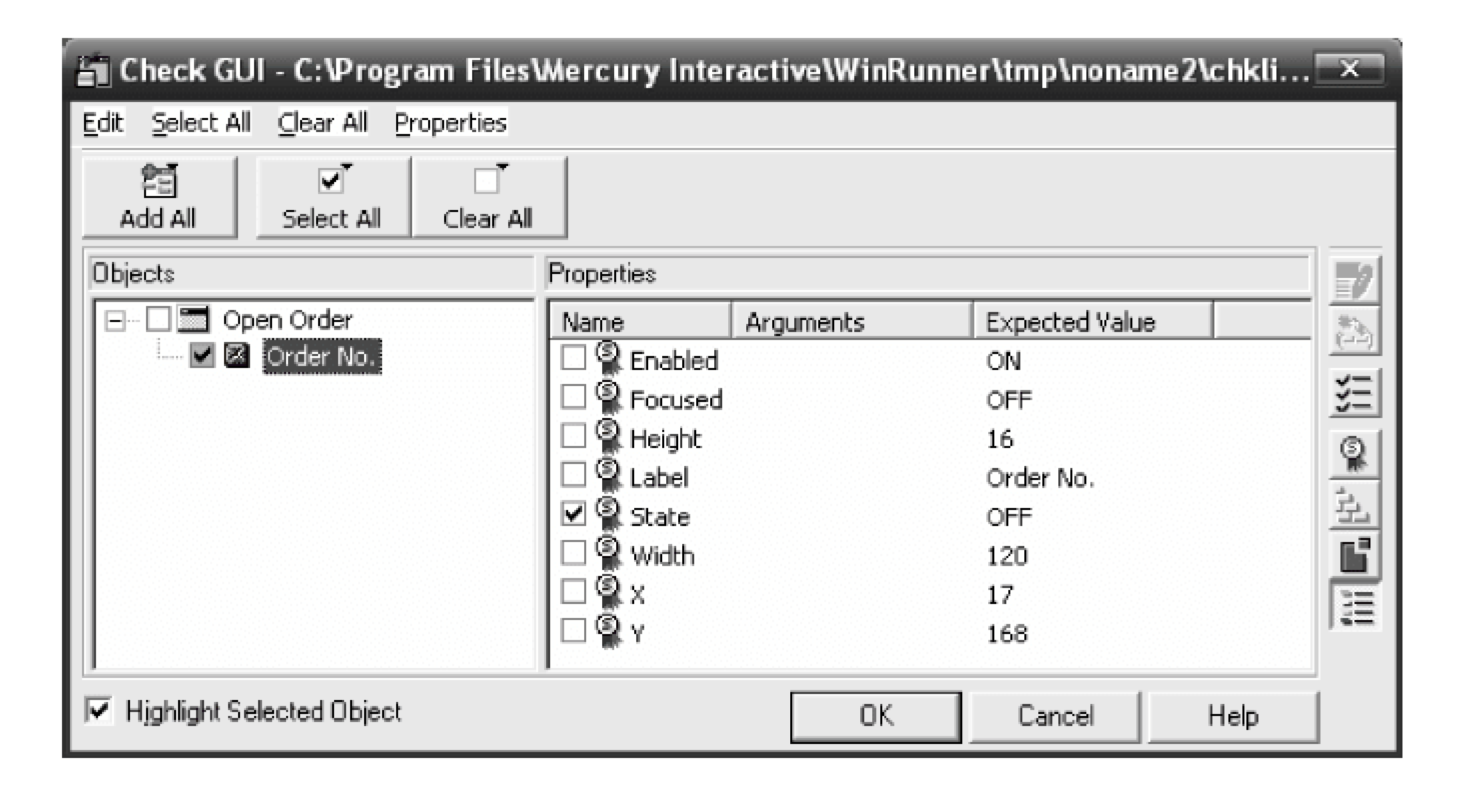


图 6.20　Check GUI 窗口

（8）对【Customer Name】复选框建立一个检查点。在 WinRunner 中选择【Insert】→【GUI Checkpoint】→【For Object/Window】命令，或单击使用者工具栏上的按钮。双击【Customer Name】复选框，则【Check GUI】对话窗口会开启并显示选取的 GUI 对象，以及此 GUI 对象可供检查的属性。如果是单击，则【Check GUI】对话窗口将不会开启，且 WinRunner 会直接以【State】属性当成检查点要检查的属性，并插入检查点。

选中【State】与【Enabled】属性，其预期值分别为 OFF 与 ON。单击【OK】按钮，WinRunner 会在测试脚本中插入 obj_check_gui 检查点。

（9）单击【OK】按钮开启订单。

（10）停止录制测试脚本。

（11）储存测试脚本为 ex4。

执行 ex4 测试脚本，以验证测试脚本可以正常执行，其步骤如下。

（1）确认 WinRunner 与 Flight Reservation 的主窗口都已经开启。

（2）开启 ex4 测试脚本。

（3）确认工具栏上显示 Verify 模式。

（4）选择 Run From Top。选择【Test】→【Run From Top】命令或单击工具栏上的 From Top 按钮，则 Run Test 窗口将会开启，接受预设 res1 的执行名称，确认已选中【Display test results at the end of run】复选框，单击【OK】按钮开始执行测试。

（5）检视测试结果。当执行结束，WinRunner 会自动开启测试执行结果。每个【end GUI checkpoint】都应该是绿色的文字，表示检查点是通过的。双击最后一个【end GUI checkpoint】，会开启【GUI Checkpoint Results】窗口，显示此检查点的测试结果。

（6）关闭【Test Results】窗口。

6.7　图像检查点

1. 如何检查应用程序的图像

如果测试人员想要检查应用程序中的图像，WinRunner 提供图像的检查点（bitmap checkpoint），以像素（pixel）的方式比对图像。

WinRunner 提供下面 3 种方式建立图像检查点。

• 屏幕区域(screen area)：以鼠标拖拉方式决定图像检查点的区域。
• 窗口：以整个窗口作为图像检查点的区域。
• GUI 物件：以整个 GUI 对象作为图像检查点的区域。

下面实例是以屏幕区域(screen area)方式建立图像检查点。

WinRunner 会直接获取方框区域部分并储存成预期值，然后在测试脚本中插入 obj_check_bitmap 或是 win_check_bitmap 指令。

当执行测试脚本时，WinRunner 会比对执行时的图像与预期的图像，将结果显示在【Test Results】窗口中，如果不一致，在【Test Results】窗口提供检视差异的部分。

2. 建立图像检查点

实例 4 将通过以图像检查点方式，测试传真订单(Fax Order)对话窗口中的签名功能。

实例 4：

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 开启 Flight Reservation 并登录。

(3) 开始以 Context Sensitive 模式录制测试脚本。

(4) 开启订单。在 Flight Reservation 选取【File】→【Open Order】命令，选中【Order No.】复选框，输入“4”然后单击【OK】按钮。

(5) 传真订单。在 Flight Reservation 选取【File】→【Fax Order】命令。

(6) 输入传真号码。在【Fax Number】文本框中输入 10 位数字，不需要输入括号与横线，如 0288303456。

(7) 移动传真订单窗口。将窗口移动到新的位置。

(8) 切换到 Analog 录制模式。按键盘上的 F2 键或单击工具栏上的 Record 按钮，此时录制模式将从 Context Sensitive 模式切换到 Analog 模式。

(9) 在【Agent Signature】中签下名字。

(10) 切换到 Context Sensitive 模式。按键盘上的 F2 键或单击工具栏上的 Record 按钮，此时录制模式会从 Analog 模式切换回 Context Sensitive 模式。

(11) 建立图像检查点检查签名。选取【Insert】→【Bitmap Checkpoint】→【For Object/Window】命令，或单击使用者工具栏上的按钮，单击【Agent Signature】选项，WinRunner 会获取【Agent Signature】的图像，并且在测试脚本中插入 obj_check_bitmap 指令。

(12) 清除签名。单击【Clear Signature】按钮，清除签名。

(13) 再建立图像检查点。选取【Insert】→【Bitmap Checkpoint】→【For Object/Window】命令，或单击使用者工具栏上的按钮，单击【Agent Signature】选项，WinRunner 会获取【Agent Signature】的图像，并且在测试脚本中插入 obj_check_bitmap 指令。

(14) 关闭传真订单窗口。单击【Cancel】按钮关闭传真订单窗口。

(15) 停止录制。

(16) 储存测试脚本为 ex5。

(17) 如果使用 Global GUI Map File 模式要将 GUI Map File 存档。在 WinRunner 中选择【Tools】→【GUI Map Editor】命令。在 GUI Map Editor 中选择【View】→【GUI Files】命令，然后选取【File】→【Save】命令。

接下来可以检视 ex5 测试脚本的预期结果，其步骤如下。

(1) 开启 WinRunner 测试结果窗口。选择【Tools】→【Test Results】或单击工具栏上的按钮,开启测试结果窗口。

(2) 检视 WinRunner 获取的图像。双击第一个 capture bitmap 事件,或单击工具栏上的按钮,开启第一个获取的图像。双击第二个 capture bitmap 事件,或单击工具栏上的按钮,开启第二个获取的图像。

(3) 关闭测试结果窗口。在测试结果窗口选择【File】→【Exit】命令关闭测试结果窗口。

以下是建立图像检查点时的建议。

(1) 如果要以屏幕区域(screen area)建立图像检查点,选择【Insert】→【Bitmap Checkpoint】→【For Screen Area】命令,或单击工具栏按钮,则鼠标会变成十字光标,然后再以拖拉方式框出要比对的区域。WinRunner 会插入 win_check_bitmap 指令,此指令参数包含屏幕区域的 x 坐标、y 坐标、宽度以及高度。

(2) 如果打算在深夜或无人时执行测试,可以设定当检查点不一致时,WinRunner 不要显示信息以免中断测试的执行。选择【Tools】→【General Options】→【Run】→【Settings】命令,清除【Break when verification fails】选项,则在测试执行过程中就不会被检查点不一致的信息中断了,如图 6.21 所示。

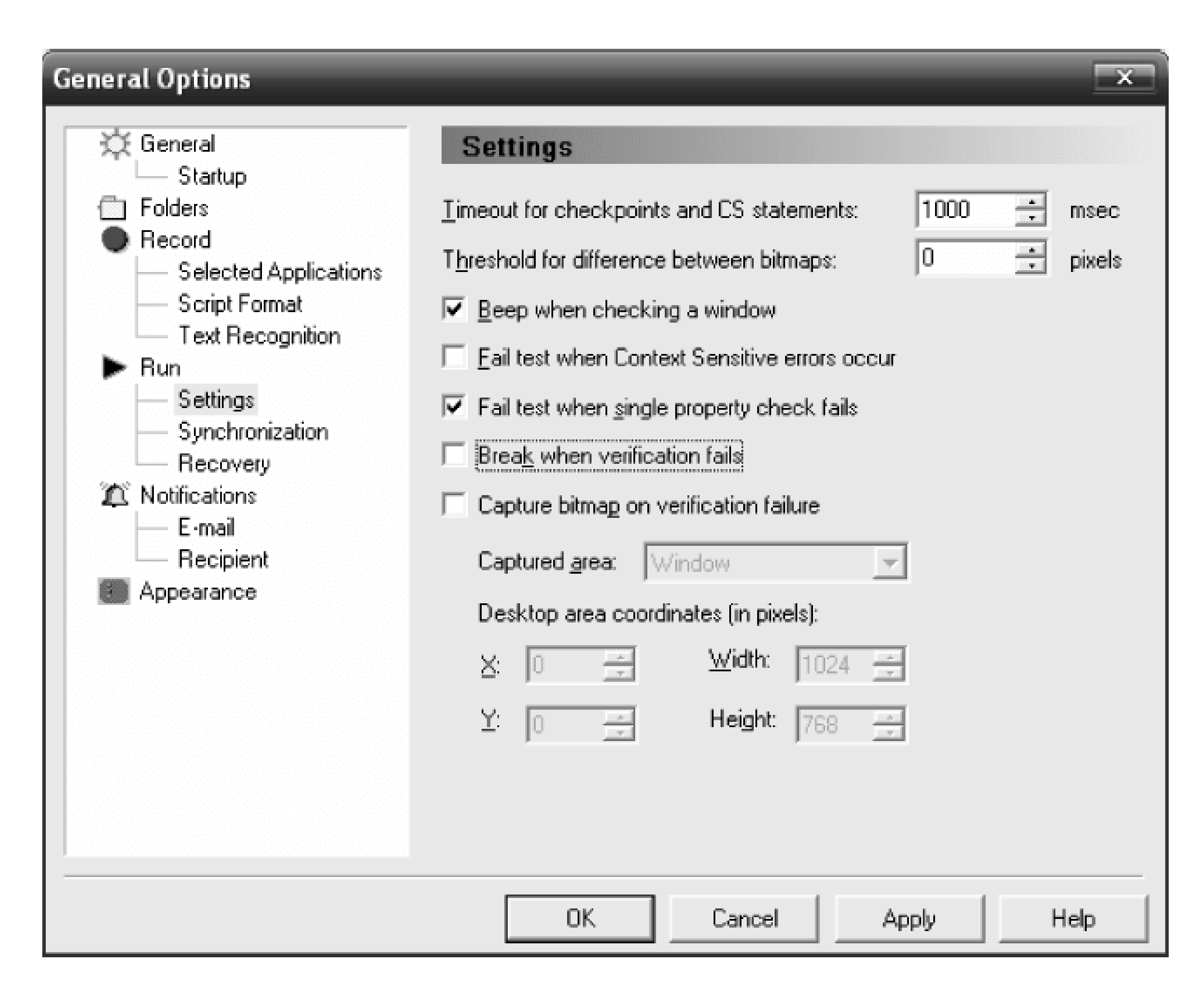


图 6.21 General Options 窗口

(3) 当执行含有图像检查点的测试脚本时,要确认屏幕显示设定与显示卡驱动程序与当初测试脚本建立时一样。如果不一样,可能会影响图像检查点的正确性。

(4) 假如测试人员想要更新图像检查点的预期值,请以 Update 模式执行一次测试脚本,则 WinRunner 会以执行当时获取到的图像,覆盖原本的预期值,成为新的预期值。

6.8 编辑测试脚本

当测试人员在录制测试脚本时,对应用程序的所有操作,不管是单击按钮还是键盘的输入,WinRunner 都会产生测试脚本,每一行的测试脚本称为 TSL(Test Script Language)。

除了以录制的方式产生测试脚本之外，TSL 还内建了许多函数，可以依照需求很弹性的应用这些功能强大的函数。除此之外，WinRunner 还提供可视化工具函数产生器（Function Generator），帮助测试人员在测试脚本中快速插入函数。

函数产生器（Function Generator）提供两种使用方式：测试人员可以直接点选 GUI 对象，让 WinRunner 建议合适的函数，然后再把函数加入测试脚本中。测试人员可以直接依照分类，从函数清单中挑选要使用的函数。

除了使用函数外，TSL 也提供一般程序语言具备的元素，如条件判断（condition）、循环（loop）、表达式（arithmetic operator）。

1. 录制基本测试脚本

实例 5：

从开启订单开始录制。

（1）开启 WinRunner 并加载 GUI Map File。

（2）开启 Flight Reservation 并登录。

（3）开始以 Context Sensitive 模式录制测试脚本。

（4）开启订单。在 Flight Reservation 选取【File】→【Open Order】，选中【Order No.】命令，输入“3”然后单击【OK】按钮。

（5）传真订单。在 Flight Reservation 选取【File】→【Fax Order】命令。

（6）单击【Cancel】按钮关闭传真订单窗口。

（7）停止录制测试脚本。

（8）储存测试脚本为 ex6。

2. 使用函数产生器（Function Generator）在测试脚本中插入函数

通过加入函数的方式，取得传真订单窗口上的 # Tickets、Ticket Price、Total 各字段的值。

（1）在“button_press(“Cancel”)；”脚本前插入一行空白。

（2）开启传真订单窗口。在 Flight Reservation 选取【File】→【Fax Order】命令。

（3）取得 # Tickets 字段的值。选择【Insert】→【Function】→【For Object/Window】命令，或单击使用者工具栏上的按钮，选择“# Tickets”的值。

函数产生器会开启并建议使用 edit_get_text 函数，如图 6.22 所示。

这个 edit_get_text 函数会取得 # Tickets 字段的值，并储存到变量中。变量的预设名称为 text。请直接将变量名称 text 改成 tickets，然后单击【Paste】按钮将函数插入测试脚本中。

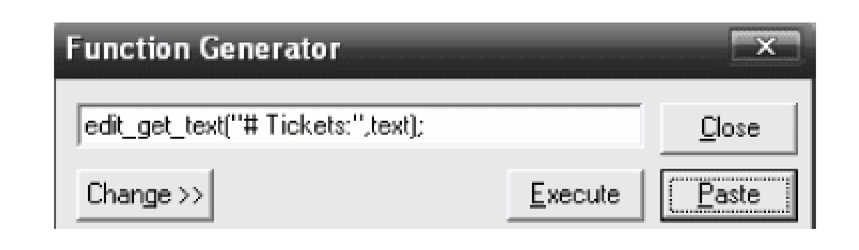


图 6.22　Function Generator 窗口

```
edit_get_text("#Tickets:",tickets);
```

（4）取得 Ticket Price 字段的值。选择【Insert】→【Function】→【For Object/Window】命令，或单击使用者工具栏上的按钮。函数产生器会开启并建议使用 edit_get_text 函数。将变量名称 text 改成 price，然后单击【Paste】按钮将函数插入测试脚本中。

```
edit_get_text("Ticket Price:",price);
```

(5) 取得 Total 字段的值。选择【Insert】→【Function】→【For Object/Window】命令,或单击使用者工具栏上的按钮。函数产生器会开启并建议使用 edit_get_text 函数。将变量名称 text 改成 total,然后单击【Paste】按钮将函数插入测试脚本中。

```
edit_get_text("Total:",total);
```

(6) 单击【Cancel】按钮关闭传真订单窗口。

(7) 储存测试脚本。

接下来将在测试脚本中加上 if/else 的判断式,如此测试脚本便可以通过计算方式判断测试是否通过。

(1) 将光标置于最后一个 edit_get_text 脚本的下一行。

(2) 加上下列的脚本。

```
if(tickets * price == total)
tl_step("total",0,"Total is correct.");
    else
tl_step("total",1,"Total is incorrect.");
```

(3) 加上批注。

在 if 脚本前加上一行空白,然后选取【Edit】→【Comment】命令,然后在"#"后加上批注。

```
# check that the total ticket price is calculated correctly.
if(tickets * price == total)
tl_step("total",0,"Total is correct.");
    else
tl_step("total",1,"Total is incorrect.");
```

(4) 储存测试脚本。

通过加上 tl_step 函数,可以自行决定测试脚本中的某段动作是通过还是失败,进而决定整个测试脚本的执行结果是通过还是失败。

举例来说:

```
tl_step("total",1,"Total is incorrect.");
```

第一个参数"total"代表这个动作的名称。

第二个参数为"1"则 WinRunner 会判定此动作为失败,如果参数值为"0"则 WinRunner 会认定此动作为通过。

第三个参数"Total is incorrect"则是 WinRunner 针对此动作显示的信息,通过有意义的描述,帮助检视最后测试结果时,更多了解此动作代表的意义。

如果要更深入的了解 tl_step 函数的用法,请参考 WinRunner TSL Online Reference。

在修改完测试脚本后,通常会执行看运行是否顺利,是否有语法或逻辑上的错误,WinRunner 同时也提供了除错的工具。通过使用除错工具,可以进行以下操作。

- 逐行执行测试脚本

- 设定断点
- 以 Watch List 检视变数的值

以 Debug 模式执行测试脚本，测试结果会储存在 debug 目录下，每次以 Debug 模式执行测试脚本后，WinRunner 会覆写前一次 debug 的执行结果。

(1) 选取 Debug 模式。选取工具栏上的 Debug 模式。

(2) 将执行箭头放在测试脚本第一行。用鼠标在测试脚本第一行左边灰色地方点一下，会出现一个黄色小箭头。

(3) 逐行执行。选取【Debug】→【Step】命令，或单击工具栏上按钮，WinRunner 开始执行第一行测试脚本。

(4) 逐行执行完整个测试脚本。继续单击工具栏上按钮，一行一行执行完整个测试脚本。

(5) 停止执行。执行完最后一行后，单击工具栏上 Stop 按钮。

(6) 检视测试结果。当以 Debug 模式执行完测试脚本，执行结果窗口并不会自动开启。选取【Tools】→【Test Results】命令，或单击工具栏上的按钮，将会开启测试结果窗口。

(7) 关闭测试结果窗口。在测试结果窗口选取【File】→【Exit】命令。

6.9 数据驱动测试脚本

建立好测试脚本后，测试人员可能会想要用多组不同的数据去执行测试脚本。为此目的，测试人员可以将测试脚本转换成数据驱动(Data-Driven)测试脚本，并建立一个数据表提供测试所需的多组数据。

可以通过下列步骤将测试脚本转换成数据驱动测试脚本。

(1) 加上开启及关闭数据表的指令。

(2) 加上循环并读取数据表的每一组数据。

(3) 将录制的固定值与检查点的值参数化为数据表的字段值。

可以使用数据驱动精灵(Data Driver Wizard)或自己手动修改测试脚本，将测试脚本转成数据驱动测试脚本。

当执行数据驱动测试脚本时，WinRunner 会读取数据表的每一笔数据，并放入被参数化的地方，然后执行一次。每执行一次称为一个反复(iteration)，数据表有几组数据，WinRunner 就会执行几次反复。并在最后的测试结果中显示每一次反复的测试结果。

在 6.8 节中，已经建立了一个测试脚本，开启一笔订单数据，并检查单价乘上票数是否等于总金额。下面会建立一个相同的检查测试脚本，并且开启多笔订单，以验证不同的票数与单价的计算也是正确的。

将测试脚本转成数据驱动(Data-Driven)测试脚本的举例如下。

实例 6：

把 ex6 录制的测试脚本转成数据驱动测试脚本。

(1) 开启 ex6 测试脚本。启动 WinRunner，打开 ex6 测试脚本。选取【File】→【Save

As】命令将 ex6 另存成 ex7。检查 GUI Map File 是否已经加载，选择【Tools】→【GUI Map Editor】命令开启 GUI Map Editor，再选择【View】→【GUI Files】命令检查是否加载 flight4a. gui。如果 flight4a. gui 没有加载，选择【File】→【Open】命令，然后选取 flight4a. gui 后，单击【Open】按钮将其载入。

（2）执行数据驱动精灵。选取【Table】→【Data Driver Wizard】命令，数据驱动精灵的欢迎窗口会开启。单击【Next】按钮，如图 6. 23 所示。



图 6. 23　DataDriver Wizard 窗口

（3）建立数据表。在【Use a new or existing Excel table】文本框中输入“ex7. xls，”数据驱动精灵会自动建立一个 Excel 档案，并储存在测试脚本的目录下。

（4）指定数据表的变量名称。【Assign a name to the variable】使用默认值 table 为数据表的变量名称。在测试脚本的开头，会以数据表的变量来取代数据表的完整路径与文件名，这样，当测试人员想要用其他的数据表来取代原本的测试数据时，只要修改此变量的值就可以了。

（5）设定参数化选项。【Add statements to create a data-driven test】选项表示由数据驱动精灵自动将转成数据驱动测试脚本的指令加到测试脚本中，预设是选中的。

【Parameterize the test】　此选项表示要做参数化，预设是选中的。

【Line by line】　WinRunner 会显示可以做参数化的脚本，预设是选中的。

单击【Next】按钮。

（6）选择要被参数化的值。第一个显示要参数化的测试脚本为“button_set ("Order No. ",ON)；”，这行脚本是选中【Order No. 】复选框，不是要作参数化的测试脚本，选中【Do not replace this data】复选框，单击【Next】按钮。

第二个显示要参数化的测试脚本为“edit_set ("Edit","3")；”，这行脚本是在【Order No. 】字段中输入“3”，就是要做参数化的脚本，此时可以看到在【Argument to be replaced】字段中显示要被参数化的资料为“3”。

在【Replace the selected value with data from：】下选取【A new column】项，并在字段中输入“Order_Num”，则数据驱动精灵会在 ex7. xls 中新增一栏 Order_Num 字段，且第一笔数据为被参数化的资料 3，单击【Next】按钮。

（7）完成。单击【Finish】按钮，数据驱动精灵将测试脚本转成如下数据驱动测试脚本。

```
table = "ex7.xls";
rc = ddt_open(table,DDT_MODE_READ);
if (rc! = E_OK && rc ! = E_FILE_OPEN)
pause("Cannot open table.");
ddt_get_row_count(table,table_RowCount);
for(table_Row = 1;table_Row<= table_RowCount; table_Row++ )
{
```

```
ddt_set_row(table,table_Row);

# Flight Reservation
set_window ("Flight Reservation",20);
menu_select_item ("File;Open Order...");

# Open Order
set_window ("Open Order",0);
button_set ("Order No.",ON);
edit_set ("Edit",ddt_val(table,"Order_Num"));
button_press ("OK");

# Flight Reservation
set_window ("Flight Reservation",3);
menu_select_item ("File;Fax Order...");

# Fax Order No. 3
set_window ("Fax Order No. 3",4);
edit_get_text("#Tickets:",tickets);
    edit_get_text("Ticket Price:",price);

# check that the total ticket price is calculated correctly.
    if(tickets * price == total)
  tl_step("total",0,"Total is correct.");
else
  tl_step("total",1,"Total is incorrect.");

button_press ("Cancel");

}
ddt_close(table);
```

将数据加入数据表的步骤如下。

（1）开启数据表。选择【Table】→【Data Table】命令开启数据表，可以看到第一栏为Order_Num，且其第一笔资料为3。

（2）加上数据。加上4笔数据，分别为1、5、7、11。

（3）储存数据表。选择【File】→【Save】命令将数据表存盘，选择【File】→【Close】命令关闭数据表。

（4）储存测试脚本。以regular expression调整测试脚本。测试脚本已经接近完成，不过在执行测试脚本之前，还是要先检查一下测试脚本是否有冲突的地方。虽然数据驱动精灵已经帮助测试人员将测试脚本中需要作参数化的值以参数取代掉了，但是数据驱动精灵并没有帮测试人员取代像对象label的值，这些固定的值可能会导致数据驱动测试脚本执行失败。

在 Flight Reservation 范例程序中，传真窗口的 label 会随着开启的订单编号而改变，所以如果执行刚刚转换成数据驱动的测试脚本，在第二次反复(iteration)时，就会出现找不到窗口的错误信息。要解决这个问题，可以通过 regular expression 来解决。所谓 regular expression 就是利用某些字符来表示特定的字符，例如用“ * ”来表示所有字符。下面将传真窗口的 label 属性修改成 regular expression，以解决找不到窗口的问题。

(1) 在 flight4a.gui 找到 Fax Order 窗口。选择【Tools】→【GUI Map Editor】命令。选取【View】→【GUI Files】命令。选择 flight4a.gui，选取 Fax Order No.3 窗口。

(2) 修改窗口 label 属性。单击【Modify】按钮，开启 Modify 窗口，如图 6.24 所示。

在【Physical Description】字段中，将 label 这一行第一个双引号后加上“!”，然后将“3”与前面的空白删除改成“ * ”号，如图 6.25 所示。

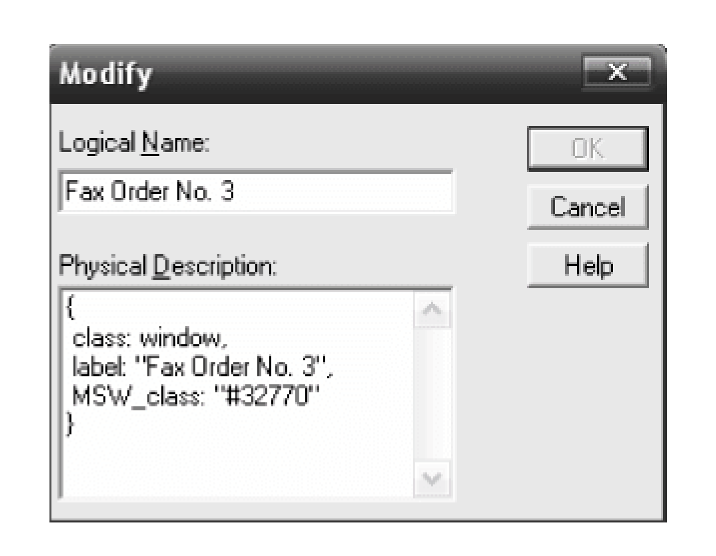


图 6.24　Modify 窗口(1)

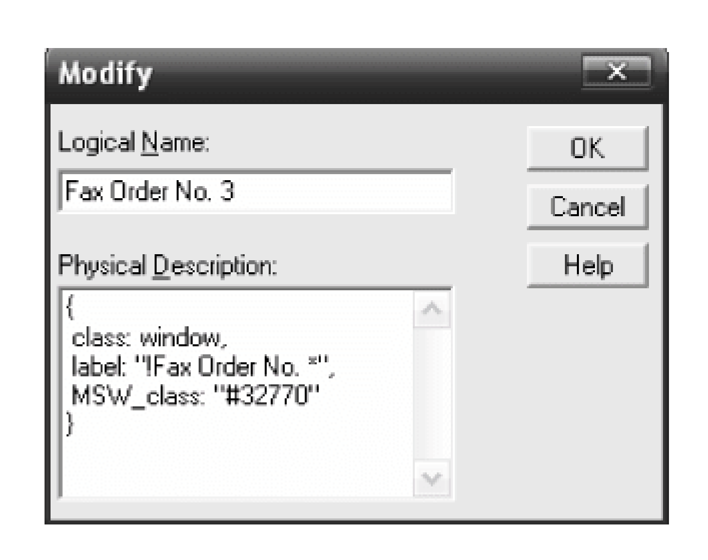


图 6.25　Modify 窗口(2)

(3) 关闭 Modify 窗口。单击【OK】按钮关闭 Modify 窗口。

(4) 如果现在使用 Global GUI Map File 模式请将 GUI Map File 存盘。

在 WinRunner 中选择【Tools】→【GUI Map Editor】命令。在 GUI Map Editor 中选择【View】→【GUI Files】命令，然后选取【File】→【Save】命令。

现在可以执行这个测试脚本了，不过在显示测试结果时的信息都是一样的。为了让测试结果也能更有意义，下面将修改测试脚本的 tl_step，使其显示的信息更有意义。

(1) 修改 tl_step。

找到第一个 tl_step 脚本：

```
tl_step("total",0,"Total is correct.");
```

修改成以下脚本：

```
tl_step("total",0,"Correct."&tickets&"tickets at $ "&price&"cost $ "&total&".");
```

同样找到第二个 tl_step 脚本：

```
tl_step("total",1,"Total is incorrect.");
```

修改成以下脚本：

```
tl_step("total",1,"Error."&tickets&"tickets at $ "&price&"does not equal $ "&total&".");
```

(2) 储存测试脚本。执行测试脚本并分析结果。

执行此测试脚本，并在测试脚本执行完成后，检视测试结果。

① 确认 Flight 4A 已经开启在桌面上。

② 单击工具栏上的执行模式为 Verify 。

③ 选取【Run From Top】项。

选择【Test】→【Run From Top】命令或单击工具栏上的 From Top 按钮，则【Run Test】窗口将会开启，接受预设 res1 的执行名称，确认选择【Display test results at the end of run】复选框，单击【OK】按钮开始执行测试。

④ 检视测试结果。

当执行完测试脚本，WinRunner 会自动开启测试结果。测试结果显示了 5 笔 tl_step 记录，而且每一笔记录都显示了票数、单价、总金额的值。

⑤ 关闭测试结果。

选取【File】→【Exit】命令关闭测试结果窗口。

⑥ 关闭 Flight Reservation。

选取【File】→【Exit】命令关闭 Flight Reservation 范例程序。

⑦ 关闭测试脚本。

选取【File】→【Close】命令关闭测试脚本。

建立数据驱动脚本时的建议如下。

(1) 可以只将测试脚本的一部分转成数据驱动测试脚本，只要在开启数据驱动精灵前，先选取要转成数据驱动的测试脚本即可。同一测试脚本中也可以包含多个数据驱动测试脚本。

(2) 可以开启 default. xls 然后储存成其他文档名，以便使用多个测试数据表。

(3) GUI 检查点、图像检查点、图像同步点、常数都可以参数化。

(4) 数据表的使用方式与 Excel 工作表相同，也可以在储存格中使用公式。

(5) 在执行数据驱动测试脚本之前，应该先检查整个测试脚本及其他部分，如 GUI 对象的属性等，看看是否有冲突的部分。以下是解决冲突的两种方案：

① 使用 regular expression 将属性变动的部分以特殊字符取代。

② 重新设定 GUI Map Configuration，将会变动的属性排除掉。

(6) 测试执行时并不需要开启数据表检视器(data table viewer)。

6.10　文字检查点

WinRunner 提供读取图像或非标准 GUI 对象上的文字的功能，并且可以手动撰写测试脚本来检查文字是否正确。

举例来说，可以通过文字检查点(text checkpoint)达到下列的目的。

- 验证某个值是否在一定范围之内
- 计算数值是否正确
- 当某个指定的文字出现在画面上时，就执行某些动作

只要指定要读取文字的区域、对象或窗口，就可以建立文字检查点。

WinRunner 会以 win_get_text 或是 obj_get_text 读取文字，并将读取到的文字储存到变量中，然后再以手动撰写测试脚本的方式，检查变量中的文字是否为预期的文字。

另外，当要验证标准的 GUI 对象(按钮、功能选单、list、edit box 等)上的文字时，建议只要使用 GUI 检查点，以省去手动撰写测试脚本的不便。

实例 7：

1. 建立测试脚本

- 开启图表并读取卖出的票数
- 新增一笔订单
- 再开启图表检查卖出的票数是否被更新
- 回报数值是否正确

2. 从应用程序读取文字

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 开启 Flight Reservation 并登录。

(3) 确认文字识别的设定。选择【Tools】→【General Options】命令，开启 General Option 窗口，选择【Record】→【Text Recognition】命令，确认【Timeout for Text Recognition】设定为合理的值(如不为 0)，默认值为 500。确认完单击【OK】按钮关闭窗口。

(4) 开始以 Context Sensitive 模式录制测试脚本。在 WinRunner 中选择【Test】→【Record-Context Sensitive】命令或单击工具栏上的 Record 按钮。

(5) 开启图表。在 Flight Reservation 中选择【Analysis】→【Graphs】命令。

(6) 读取图表上的票数。在 WinRunner 中选择【Insert】→【Get Text】→【From Screen Area】命令或单击工具栏上的按钮。此时鼠标光标会变成十字光标，以左键拖拉的方式框住票数后，再以鼠标右键结束操作，如图 6.26 所示。

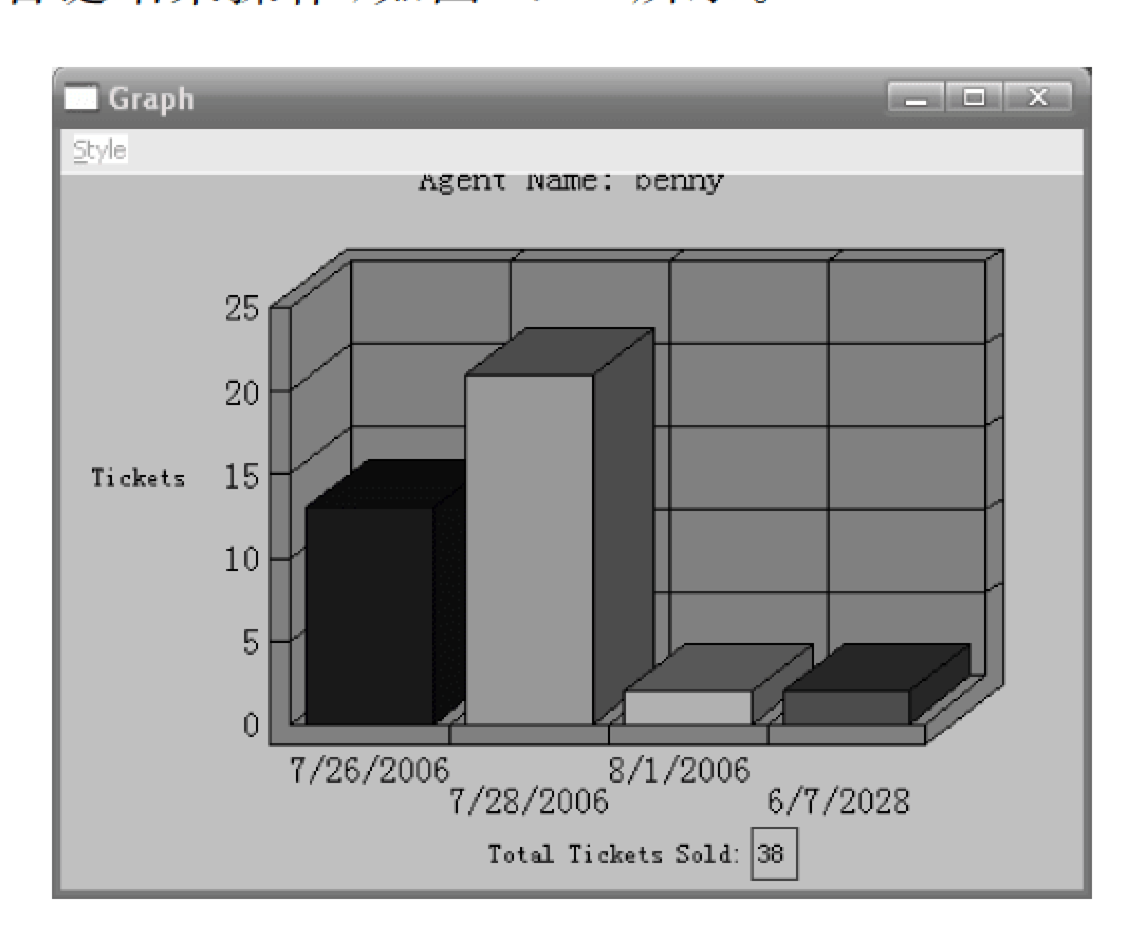


图 6.26　Graph 窗口

WinRunner 会插入 obj_get_text 指令，并且在后面加上批注文字「#38」，表示目前读取到的文字为“38”。

```
obj_get_text("GS_Drawing",text,308,291,328,314);    #38
```

(7) 关闭图表窗口。

(8) 建立新订单。在 Flight Reservation 中选取【File】→【New Order】命令。

(9) 填入航班与旅客资料。请输入以下数据：

【Date of Flight】：08/10/06(日期格式为 MM/DD/YY，日期要大于今天的日期)

【Fly From】：London

【Fly To】：San Francisco

点击【Flights...】按钮，选取一个航班

【Name】：benny

【Class】：First

【Tickets】：1

(10) 新增订单。选择【Insert Order】命令，当完成新增订单后，状态列会显示【Insert Done...】的信息。

(11) 插入同步点。选择【Insert】→【Synchronization Point】→【For Object/Window Bitmap】命令，或单击使用者工具栏上的按钮。将鼠标光标移动到【Insert Done...】的状态列上并点选，WinRunner会在测试脚本中插入一行"obj_wait_bitmap ("Insert Done...", "Img1",14);"的指令。

(12) 再开启图表。在Flight Reservation中选择【Analysis】→【Graphs】命令。

(13) 读取图表上的票数。在WinRunner中选择【Insert】→【Get Text】→【From Screen Area】命令或单击工具栏上的按钮。此时鼠标光标会变成十字光标，以左键拖拉的方式框住票数后，再以鼠标右键结束操作。WinRunner会插入obj_get_text指令，并且在后面加上批注文字"#39"，表示目前读取到的文字为"39"。

(14) 关闭图表窗口。

(15) 停止录制测试脚本。

(16) 储存测试脚本为ex8。

(17) 如果在Global GUI Map File模式下，记得储存新的GUI对象。

3. 检查文字

下面通过if/else验证当新增一笔机票订单后，图表上的票数是否有更新。

(1) 在第一个obj_get_text指令将text变量名称改成first_total。

(2) 在第二个obj_get_text指令将text变量名称改成new_total。

(3) 将光标移到测试脚本最后一行。

(4) 加入以下的测试脚本。

当new_total=first_total+1则回报检查点通过，反之则回报检查点失败。

```
  if(new_total == first_total + 1)
{
  tl_step("graph total",0,"Total is correct.");
}
else
{
      tl_step("graph total",1,"Total is incorrect.");
}
```

(5) 加上批注。在if前加上以下批注：

```
# check that graph total increments by one.
```

(6) 储存测试脚本。

4. 除错

以除错(debug)模式执行测试脚本,检查是否有语法或逻辑上的错误。如果有任何错误信息,试着去修正问题。

(1) 选取 Debug 模式。

(2) 执行测试脚本。

(3) 检视测试结果。以 Debug 模式执行完测试脚本,执行结果窗口并不会自动开启。选取【Tools】→【Test Results】命令,或单击工具栏上的按钮,将会开启测试结果窗口。

(4) 关闭测试结果窗口。在测试结果窗口选择【File】→【Exit】命令。

(5) 关闭 Flight Reservation。在 Flight Reservation 中选择【File】→【Exit】命令。

6.11 批次测试

1. 何谓批次(batch)测试

想象一下这样的情况,测试人员刚刚变更了自己的应用程序,然后要在新版的应用程序上执行所有的测试脚本。测试人员不需要一个一个单独地执行测试脚本,只需要执行一个批次测试,然后就可以去做其他事情,过段时间,屏幕上已经显示所有测试脚本的测试结果。

批次测试脚本看起来与一般的测试脚本一样,但实际上批次测试脚本与一般测试脚本有下面两个不同的地方。

(1) 批次测试脚本含有 call 指令,用来开启其他测试脚本,例如:

```
call"c:\\qa\\flights\\ex4"();
```

当批次测试执行时,WinRunner 一执行到 call 指令,便会开启并执行指定的测试脚本,当被呼叫的测试脚本执行完毕,WinRunner 便会回到批次测试继续执行下去。

(2) 在执行批次测试之前,测试人员要先选择【Tools】→【General Options】命令,在选择【Run】后选取【Run in batch mode】选项,这个选项会让 WinRunner 不再跳出信息对话窗口而中断测试的执行。例如当一个图像检查点失败时,WinRunner 不会再暂停测试执行并显示 mismatch 的信息了。

当测试人员检视测试结果时,可以看到整个批次测试的测试结果是通过还是失败,也可以看到所有被批次测试呼叫的测试,其结果是通过还是失败。

2. 建立批次测试

实例 8:

建立一个批次测试。

- 呼叫之前建立的测试脚本(ex4、ex5、ex6)。
- 执行每个被呼叫的测试脚本 3 次。

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 加上 call 指令呼叫其他测试脚本。在新开启的测试脚本中输入以下的脚本:

```
call"c:\\qa\\flights\\ex4"();
call"c:\\qa\\flights\\ex5"();
call"c:\\qa\\flights\\ex6"();
```

在测试脚本中,请将 c:\\qa\\flights 换成测试脚本存放的路径。

注意：在 WinRunner 的测试脚本中用双斜线"\\"取代一般档案路径的斜线"\"。

(3) 加上 loop 循环。为了执行 3 次所有被呼叫的测试脚本，请加上以下的 loop 循环：

```
for(i = 0;i<3;i++)
{
    call"c:\\qa\flights\\ex4"();
    call"c:\\qa\flights\\ex5"();
    call"c:\\qa\flights\\ex6"();
}
```

(4) 设定以批次模式执行。选择【Tools】→【General Options】→【Run】命令，选取【Run in batch mode】选项，然后单击【OK】按钮。

(5) 储存批次测试脚本

3. 建立批次测试脚本时的建议

(1) 测试人员可以设定测试脚本的搜寻路径，WinRunner 会自动到设定的路径下搜寻被呼叫的测试脚本，这样在批次测试中呼叫其他测试脚本时，就不需要输入完整的测试脚本的路径，而只要输入测试脚本名称就可以了。

选择【Tools】→【General Options】→【Folders】命令，在【Search path for called tests：】中输入测试脚本的搜寻路径，然后单击 + 按钮，即完成设定测试脚本的搜寻路径。设定完成后，在批次测试脚本中呼叫其他测试脚本时，就不需要输入完整的路径，而只要输入测试脚本名称就可以了。

(2) 在选取【Tools】→【General Options】→【Run】命令时，选中【Run in batch mode】选项，否则当批次脚本执行过程中如果有任何错误发生，将导致测试执行中断。

6.12 维护测试脚本

想象一下这样的情况：测试人员花了几个礼拜的时间，建立了一整套涵盖应用程序所有功能的测试脚本。开发团队为了改善使用者接口，修改了一些 GUI 对象，也新增了一些 GUI 对象，甚至删除了一些 GUI 对象，并且发行了新版本。而测试人员要如何用现有的测试脚本来测试新版的应用程序呢？

面对这样的情况，WinRunner 提供了一个非常方便的解决方案：GUI Map。通过更新 GUI Map，WinRunner 就可以识别这些被新增、修改的 GUI 对象，就不需要手动去修改现有的测试脚本了。

在 GUI Map 中记录了应用程序中 GUI 对象的描述(descriptions)，其内容由以下两个部分组成。

(1) 逻辑名称(logic name)：一个简短且直接的名称，用来代表 GUI 对象，可以看到这个名称出现在测试脚本中，例如：

```
button_press("Insert Order");
```

【Insert Order】就是某个 GUI 对象的逻辑名称。

(2) 实体描述(physical description)：一组可用来唯一识别 GUI 对象的属性，例如：

```
{
  class:push_button,
  label:"insert Order"
}
```

表示这个 GUI 对象是属于【push_button】类别，也就是一个按钮，且按钮上的卷标(label)为【Insert Order】。

在执行测试脚本时，当 WinRunner 读取到一个 GUI 对象的逻辑名称后，WinRunner 会到 GUI Map 中寻找这个 GUI 对象实体描述，然后以这些属性，在应用程序上找到拥有这些属性的 GUI 对象。所以当应用程序上的 GUI 对象有变更，就必须在 GUI Map 中修改此 GUI 对象的实体描述，如此一来，WinRunner 就可以识别此 GUI 对象了。

实例 9：

(1) 在 GUI Map 中编辑 GUI 对象的属性。

(2) 新增一个 GUI 物件到 GUI Map 中。

(3) 使用执行精灵(Run wizard)自动侦测使用者界面的变动，并自动更新 GUI Map。

1. 在 GUI Map 中编辑 GUI 对象的属性

假设在新版本的 Flight Reservation 应用程序中，原本的【Insert Order】按钮已经修改成【Insert】按钮，为了要让有用到【Insert Order】按钮的测试脚本可以继续被使用，必须在 GUI Map 中修改【Insert Order】按钮的卷标。

(1) 开启 WinRunner 并加载 GUI Map File。

(2) 开启 GUI Map Editor。选取【Tools】→【GUI Map Editor】命令，开启 GUI Map Editor。

在 GUI Map Editor 选取【View】→【GUI Map】，则【Windows/Objects】会列出目前 GUI Map 的内容，每个 GUI 对象会根据其类别(class)以不同的图标显示，并显示 GUI 对象的逻辑名称。

(3) 找到【Insert Order】按钮。在【GUI Map Editor】中选择【View】→【Collapse Objects Tree】命令，以便只检视窗口。双击【Flight Reservation】窗口，【Flight Reservation】窗口会展开并显示属于【Flight Reservation】窗口的所有 GUI 对象，找到【Insert Order】按钮。

(4) 检视【Insert Order】按钮的实体描述。选择【Insert Order】按钮，在 GUI Map Editor 下方会显示【Insert Order】按钮的实体描述。

(5) 修改【Insert Order】按钮的实体描述。单击【Modify】按钮或双击【Insert Order】按钮，会开启 Modify 窗口，并显示【Insert Order】按钮的逻辑名称与实体描述。在【Physical Description】中将 label 属性从【Insert Order】改成【Insert】。单击【OK】按钮，储存修改。

(6) 关闭 GUI Map Editor。选择【File】→【Save】储存 GUI Map，然后选择【File】→【Exit】关闭 GUI Map Editor。

2. 新增 GUI 物件到 GUI Map

当应用程序新增 GUI 对象时，只要以 GUI Map Editor 的学习(learn)功能，就可以将新增的 GUI 对象加到 GUI Map 中，不需要再执行 RapidTest Wizard。GUI Map Editor 可以一次学习一个 GUI 对象或是一个窗口中的所有 GUI 对象。

3. 让 WinRunner 学习 Flight Reservation 登录窗口

(1) 开启 Flight Reservation 并登录。执行【开始】→【程序】→【WinRunner】→【Sample

Applications】→【Flight 4A】命令。

(2) 开启 GUI Map Editor。在 WinRunner 选取【Tools】→【GUI Map Editor】命令，当 GUI Map Editor 开启后选取【View】→【GUI Files】命令。

(3) 学习登录窗口的所有 GUI 对象。单击【Learn】按钮，此时鼠标光标会变成，选择登录窗口的标题列，则 WinRunner 会跳出一个信息窗口，询问是否要学习窗口中所有的 GUI 对象。单击【Yes】按钮后，注意看 WinRunner 如何将窗口中的 GUI 对象学习下来的。

(4) 储存 GUI Map。选取【File】→【Save】命令储存 GUI Map，单击【OK】按钮将新的窗口与 GUI 对象储存到 flight4a. gui 中。

(5) 关闭 GUI Map Editor。选择【File】→【Exit】命令，关闭 GUI Map Editor。

(6) 关闭登录窗口。在登录窗口中单击【Cancel】按钮。

4. 使用执行精灵(Run wizard)自动更新 GUI Map

在测试执行过程中，假如 WinRunner 在应用程序上无法找到测试脚本中所使用的 GUI 对象，就会自动开启执行精灵(Run wizard)。通过执行精灵(Run wizard)，可以让 WinRunne 自动重新识别找不到的 GUI 对象或是新增 GUI 对象。

举例来说，当 WinRunner 在 Flight Reservation 执行到单选【Insert Order】按钮的测试脚本：

```
button_press("Insert Order");
```

假设这个按钮的卷标(label)已经从【Insert Order】改成【Insert】。这时执行精灵(Run wizard)会自动开启，并提醒 WinRunner 找不到【Insert Order】按钮。然后单击按钮，并重新单击【Insert】按钮。这时执行精灵(Run wizard)会建议解决方案，单击【OK】按钮，则执行精灵(Run wizard)会自动更新【Insert Order】按钮的实体描述，并且从中断的地方继续执行下去。

(1) 开启 GUI Map Editor。在 WinRunner 中选择【Tools】→【GUI Map Editor】命令，当 GUI Map Editor 开启后选择【View】→【GUI Files】命令，并在【GUI Files】中选取 flight4a. gui。

(2) 从 GUI Map Editor 中删除【Fly From】清单对象。选择位于【Flight Reservation】窗口下的【Fly From】清单对象，并单击【Delete】按钮，删除后存档并关闭 GUI Map Editor。

(3) 开启 Flight 4A 并登录。

(4) 开启 ex3 测试脚本并执行。

注意：当 WinRunner 执行到下面这一行测试脚本时会发生什么。

```
list_select_item("Fly From:","Los Angeles");
```

(5) 依照执行精灵(Run wizard)的指示将【Fly From】清单对象加到 GUI Map，由于【Fly From】清单对象已经被删除，所以执行精灵(Run wizard)会开启，同样单击按钮，并重新选择【Fly From】清单对象，然后单击【OK】按钮，则执行精灵(Run wizard)会自动将【Fly From】清单对象加到 GUI Map 中。

另外由于之前变更了【Insert Order】按钮的实体描述，同样也使用执行精灵(Run wizard)将【Insert Order】按钮的实体描述改回来，则 WinRunner 继续完成测试脚本的执行。

(6) 储存 GUI Map。选取【File】→【Save】命令储存 GUI Map，单击【OK】按钮将新的窗口与 GUI 对象储存到 flight4a. gui 中。

(7) 关闭 GUI Map Editor。选择【File】→【Exit】命令关闭 GUI Map Editor。

(8) 关闭 Flight Reservation。选择【File】→【Exit】命令关闭 Flight Reservation。

6.13 WinRunner 测试实例

通过前面的学习,掌握了 WinRunner 的基本使用,现在利用它对 Windows 的计算器软件中加法功能进行自动化测试,计算器的操作界面如图 6.27 所示。



图 6.27 计算器用户界面

该测试的用例设计比较简单,运用第 3 章介绍的黑盒测试方法,首先对输入数据进行等价类划分。由于计算器的运算数据输入是通过单击图 6.27 中预先设定的界面按钮来实现,而不是由用户通过键盘自由录入,所以不会出现字符、空格等无效的非数值型数据,所以在进行等价类划分时对于计算器加数非空的情况下可以只考虑数值型数据即可。于是我们可以得到负数、0、正数这 3 个有效等价类,其中负数和正数又可以进一步细分为负小数、负整数和正小数、正整数。结合等价类划分和边界值分析方法,并参考计算器软件帮助说明,首先进行计算器加法功能测试用例的设计。

此处采用如表 6.3 所示的测试用例模板书写加法功能测试用例。当然不同的公司可能会有不同的测试用例书写模板,虽然风格和样式会有所区别,但它们本质上都是一样的,都包括了测试用例的基本要素,如:测试环境、操作步骤、输入数据和期望结果等。

表 6.3 计算器加法功能测试用例

项目名称	Windows 自带计算器		**程序版本**	5.1
测试环境	硬件:CPU 赛扬 2.4 G,RAM 内存 256 M,硬盘剩余空间 10 G			
	软件:Windows XP			
编制人	zxm		**编制时间**	2007.11.1
功能模块名	加法运算			
功能特性	实现 2 个数的加法运算			
测试目的	验证功能的正确性和容错性			
预置条件	选择开始菜单中的"程序/附件/计算器"选项,运行计算器程序			
参考信息	无		**特殊规程说明**	无
用例编号	**测试步骤**	**输入数据**	**预期结果**	**测试结果**
01	依次点击"1""0""0""+""1""0""1""="按钮	100,101	201	
02	依次点击"0""+""0""="按钮	0,0	0	
03	依次点击"−""1""+""1""="按钮	−1,1	0	
04	依次点击"−""1""0""0""+""−""1""0""1""="按钮	−100,−101	−201	

05	依次点击“8”“0”“+”“=”按钮	80，无	160	
06	依次点击“+”“8”“0”“=”按钮	无，80	80	
07	依次点击“9”“9”…(重复点击“9”到无法增加为止)“+”“=”按钮	999…(允许输入的最大正数,长度为 32)，无	2 * 10^{32}(由于计算器精度限制而进行四舍五入处理)	
08	依次点击“-”“9”“9”…(重复点击“9”到无法增加为止)“+”“=”按钮	-999…(允许输入的最小负数,长度为 32)，无	-2 * 10^{32}(由于计算器精度限制而进行四舍五入处理)	
09	依次点击“0”“.”“6”“+”“0”“.”“5”“=”按钮	0.6，0.5	1.1	
10	依次点击“-”“0”“.”“6”“+”“-”“0”“.”“5”“=”按钮	-0.6，-0.5	-1.1	
11	依次点击“+”“=”按钮	无，无	0	

根据上面测试用例的设计开展测试的执行工作,具体的测试执行步骤如下。

(1) 打开 Windows 的计算器软件;

(2) 对数据录入显示框清零(点击“C”按钮);

(3) 录入加数 additive1(点击数字按钮);

(4) 点击“+”按钮;

(5) 录入加数 additive2(点击数字按钮);

(6) 点击“=”按钮;

(7) 检查运算结果是否正确;

(8) 循环执行(2)~(7)步直到每组测试数据均执行完毕;

(9) 报告测试结果。

在表 6.2 中设计了 11 个用例,于是上面的执行步骤就要循环重复 11 次。如果采用人工测试,操作步骤简单而繁琐,而且当要进行回归测试时,重复工作量更大。因此,完全可以考虑借助工具来实现上述测试的自动化。因为是功能测试,所以可选用本章介绍的功能测试工具 WinRunner 来实现。

首先分别启动 WinRunner 和计算器,并对两者窗口位置进行调整以使其不重叠。然后打开 WinRunner 的 GUI Map Editor 对“计算器”窗体中的对象进行学习,并把学到的信息保存在 GUI Map 文件“计算器.gui”中。接下来在 Context Sensitive 模式下录制在“计算器”上进行的一次加法运算操作过程,从而得到一段相应的测试脚本,再在此脚本基础上对不适合要求的地方进行手动修改。这里主要是完成数据启动测试脚本和获取运行结果并进行判断报告的脚本修改工作,从而得到如下所示的自动测试程序。通常编写自动测试程序

都可以采用这种方式,不必从头一句一句编写,这样可提高编程效率。

```
#加载 GUI Map 文件并激活计算器窗口
    GUI_load("C:\\Documents and Settings\\gzg\\My Documents\\mywrtest\\计算器\\计算器.gui");
    set_window("计算器");

#打开存放测试数据的文件"data1.txt"
table = "data1.txt";
rc = ddt_open(table,DDT_MODE_READ);
if(rc! = OK && rc! = E_FILE_OPEN)
{
  tl_step("open file",1,"open data file is failed.");
  texit;
}

#循环读取数据文件中的每组数据执行测试
ddt_get_row_count(table,table_RowCount);
for (table_Row = 1;table_Row<= table_RowCount;table_Row++ )
{
  ddt_set_row(table,table_Row);
  button_press("C");
  additive1 = ddt_val(table,"additive1");
  additive2 = ddt_val(table,"additive2");
  for (i = 1;i<= length(additive1);i++ )
   button_press(substr(additive1,i,1));
  button_press (" + ");
  for (i = 1;i<= length(additive2);i++ )
    button_press(substr(additive2,i,1));
  button_press (" = ");

  edit_get_text("Edit",result);

  #删除 result 串的首尾空格与尾部小数点
  len = length(result);
  while (len>0)
if (substr(result,len,1) == " " || substr(result,len,1) == ".")
len-- ;
else
break;
  i = 1;
  while (i<len)
if (substr(result,i,1) == " ")
i++ ;
else
```

```
break;
  result = substr(result,i,len);

  #将运行结果与预期结果进行比较判断
  if ((additive1 + additive2) == result)
    tl_step("testcase"&table_Row,0,"the result is "&(additive1 +
                additive2)&" and "&result&",correct.");
  else
    tl_step("testcase"&table_Row,1,"the result is "&(additive1 +
                additive2)&" and "&result&",incorrect.");
}
ddt_close(table);
```

完成上面的自动测试程序编写后，还要记住将测试数据加入数据表。按照前面介绍的方法，在 WinRunner 中选择【Table】→【Data Table】命令，然后在预先创建的文件"data1.txt"的相应路径下找到它并选中打开，就可以开启数据表进行测试数据的输入。因为上面程序中对变量的命名规则，所以首先将数据表中第一列和第二列的列名修改为"additive1"和"additive2"，再依次录入预先设计好的每组测试数据，并进行保存。

这时就可以单击【Run From Top】运行上面的自动测试程序了。运行完毕后，可以通过窗口【WinRunner Test Results】显示的信息分析测试结果。

图 6.28 是上述自动测试脚本的执行结果窗口，从其中显示的信息可以很清楚地看到当前测试是否成功通过。如果不成功，那么是哪一组测试数据出现了问题。在本例中，设计的测试数据均成功执行。上面窗口中的显示信息也可以通过修改脚本加以改变，以使其满足不同的信息显示需要。

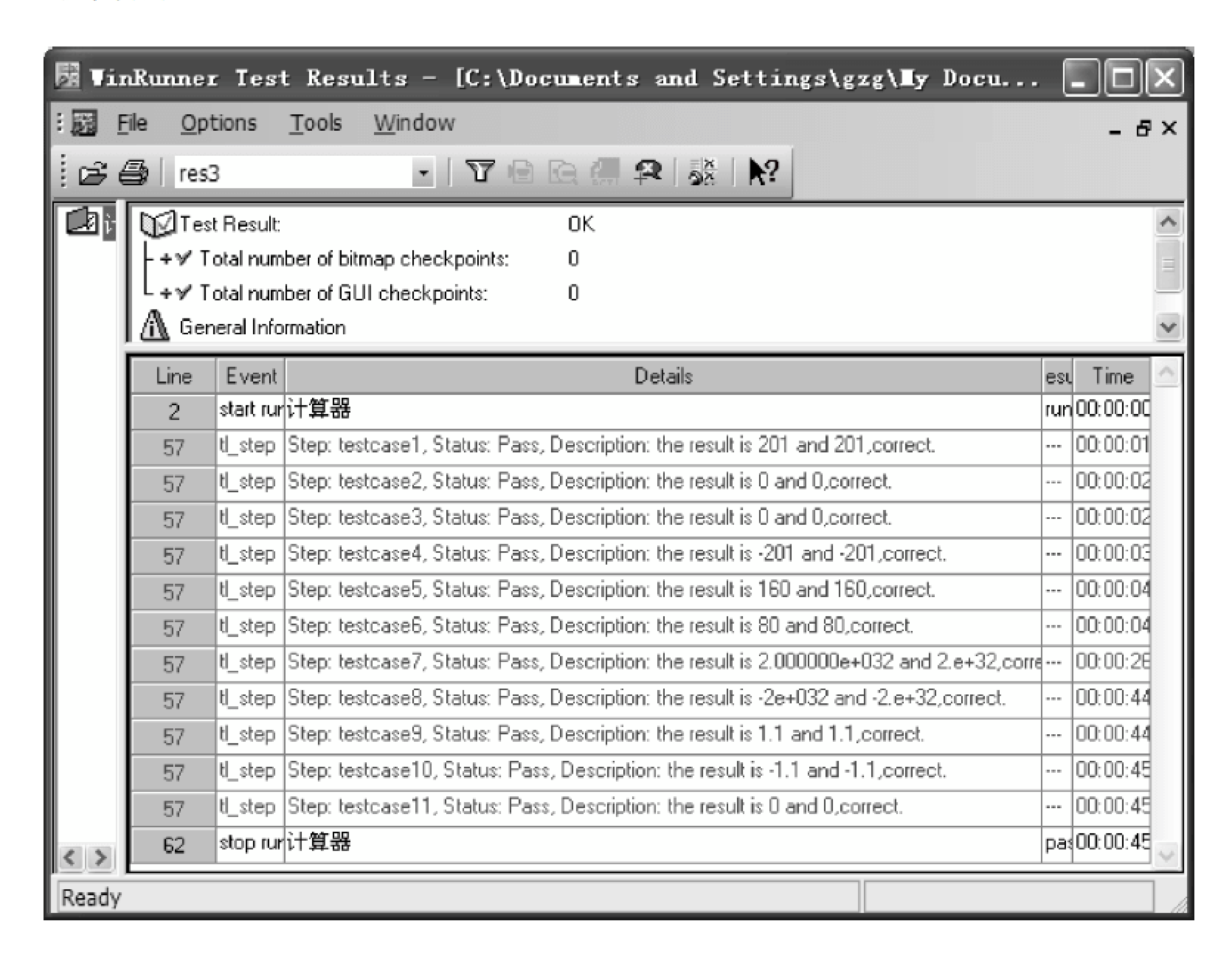


图 6.28　计算器加法测试结果

练　习　题

1. 列举几种 WR 学习软件 GUI 的不同方式。

2. 分别简述 WR 中同步点和检查点的作用。

3. 比较 WinRunner 中 GUI Map File per Test 和 Global GUI Map File 两种模式的区别。

4. 简述利用 WinRunner 进行测试的过程可分为哪几个阶段，即操作步骤是什么？

5. 给出 WinRunner 中将测试脚本转换为数据驱动测试脚本的一种实现步骤。

6. 仿照实例 4，在 Flight Reservation 样本软件的 Flight 4B 版本中建立 GUI 对象检查点。

7. 仿照实例 5，在 Flight Reservation 样本软件的 Flight 4B 版本中建立图像检查点。

8. 仿照实例 8，在 Flight Reservation 样本软件的 Flight 4B 版本中练习文字检查点的应用。

9. 仿照实例 8，在 Flight Reservation 样本软件的 Flight 4B 版本中执行批次测试。

10. 仿照计算器加法功能的测试，完成对 Windows 的计算器减法、乘法和除法的测试。

11. 思考利用 WR 测试网易邮箱的登录模块。

第 7 章　LoadRunner 测试工具

7.1　性能测试工具介绍

目前市场上的性能测试的工具种类很多，可以简单的划分为以下几种：负载压力测试工具、资源监控工具、故障定位工具以及调优工具。

1. 主流负载性能测试工具

负载性能测试工具的原理通常是通过录制、回放脚本、模拟多用户同时访问被测试系统制造负载，产生并记录各种性能指标，生成分析结果，从而完成性能测试的任务。

主流的负载性能测试工具有以下几种。

QA Load：Compuware 公司的 QALoad 是客户/服务器系统、企业资源配置(ERP)和电子商务应用的自动化负载测试工具。QALoad 是 QACenter 性能版的一部分，它通过可重复的、真实的测试能够彻底地度量应用的可扩展性和性能。QACenter 汇集完整的跨企业的自动测试产品，专为提高软件质量而设计。QACenter 可以在整个开发生命周期、跨越多种平台、自动执行测试任务。

SilkPerformer：一种在工业领域最高级的企业级负载测试工具。它可以模仿成千上万的用户在多协议和多计算的环境下工作。不管企业电子商务应用的规模大小及其复杂性，通过 SilkPerformer，均可以在部署前预测它的性能。可视的用户化界面、实时的性能监控和强大的管理报告有助于迅速的解决问题，例如加快产品投入市场的时间，通过最小的测试周期保证系统的可靠性，优化性能和确保应用的可扩充性。

LoadRunner：一种较高规模适应性的自动负载测试工具，它能预测系统行为、优化性能。LoadRunner 强调的是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来更快的确认和查找问题。此外，LoadRunner 能支持最宽泛的协议和技术，为特殊环境量身定做地提供解决方案。

WebRunner：RadView 公司推出的一个性能测试和分析工具，它让 Web 应用程序开发者自动执行压力测试；WebLOAD 通过模拟真实用户的操作，生成压力负载来测试 Web 的性能，用户创建的是基于 javascript 的测试脚本，称为议程 agenda，用它来模拟客户的行为，通过执行该脚本来衡量 Web 应用程序在真实环境下的性能。

免费测试工具有以下几种。

OpenSTA：开放源性能测试工具 OpenSTA(Open System Testing Architecture)，是用 C++ 语言开发的软件，可以执行分布式测试，通过简单的图表形式和分布的测试，对于 HTTP 测试提供了很好的性能，对于简单的和可靠的 HTTP 测试来说是很好的软件。

WAS(Web Application Stress Tool)：微软的工具，专门用来进行实际网站压力测试的

一套工具,用来模拟 Web 浏览器对使用 http1.0 或 1.1 标准的 Web 服务器的请求,而不用考虑 Web 服务器运行在哪种平台上。使用 WAS,可以用不同的方式产生测试脚本。与其他工具不同的地方在于,WAS 可以使用多个客户端机器测试 Web 站点,把其中一个客户机作为主客户端用于协调其他客户端的测试。

2. 资源监控工具

资源监控作为系统压力测试过程中的一个重要环节,在相关的测试工具中都有很多的集成。只是不同的工具之间监控的中间件、数据库、主机平台的能力以及方式各有差异。而这些监控工具更大程度上都依赖于被监控平台自身的数据采集能力,目前的绝大多数的监控工具基本上是直接从中间件、数据库以及主机自身提供的性能数据采集接口获取性能指标。

首先,不同的应用平台有自身的监控命令以及控制界面。比如 UNIX 主机用户可以直接使用 topas, vmstat, iostat 了解系统自身的健康工作状况。另外, weblogic 以及 websphere 平台都有自身的监控台,在上面可以了解到目前的 JVM 的大小、数据库连接池的使用情况以及目前连接的客户端数量以及请求状况等。只是这些监控方式的使用对测试人员有一定的技术储备要求,需要自己熟练掌握以上监控方式的使用。

第三方的监控工具相应地对一些系统平台的监控进行了集成。比如,LoadRunner 对目前常用的一些业务系统平台环境都提供了相应的监控入口,从而可以在并发测试的同时,对业务系统所处的测试环境进行监控,更好的分析测试数据。

但 Loadrunner 工具提供的监控方式还不是很直观,一些更直观的测试工具能在监控的同时提供相关的报警信息,类似的监控产品如 QUEST 公司提供的一整套监控解决方案包括了主机的监控、中间件平台的监控以及数据库平台的监控。QUEST 系列监控产品提供了直观的图形化界面,能让测试者尽快进入监控的角色。

3. 故障定位工具以及调优工具

技术的不断发展以及测试需求的不断提升,故障定位工具应运而生,它能更精细地对负载压力测试中暴露的问题进行故障根源分析。在目前的主流测试工具厂商中,都相应地提供了对应的产品支持。尤其是目前.NET 以及 J2EE 架构的流行,测试工具厂商纷纷在这些领域提供了相关的技术产品,比如,Loadrunner 模块中添加的诊断以及调优模块、QUEST 公司的 PerformaSure、Compuware 的 Vantage 套件以及 CA 公司收购的 Wily 的 Introscope 工具等,都在更深层次上对业务流的调用进行追踪。这些工具在中间件平台上引入探针技术,能捕获后台业务内部的调用关系,发现问题所在,为应用系统的调优提供直接的参考指南。

在数据库产品的故障定位分析上,Oracle 自身提供了强大的诊断模块,同时,Quest 公司的数据库产品也在数据库设计、开发以及上线运行维护上提供了全套的产品支持。

7.2 LoadRunner 简介

LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题,LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner,企业能最大限度地缩短测试时间,优化性能和加速应用系统的发布周期。

目前企业的网络应用环境都必须支持大量用户,网络体系架构中含各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢,系统崩溃等问题。这些都不可避免地导致公司收益的损失。Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源,无需购置额外硬件而最大限度地利用现有的 IT 资源,并确保终端用户在应用系统的各个环节中对其测试应用的质量,可靠性和可扩展性都有良好的评价。LoadRunner 是一种适用于各种体系架构的自动负载测试工具,它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统,它通过模拟实际用户的操作行为和实行实时性能监测,来更快的查找和发现问题。此外,LoadRunner 能支持广泛的协议和技术,为特殊环境提供特殊的解决方案。

7.2.1 LoadRunner 的基本原理

LoadRunner 启动以后,在任务栏会有一个 Agent 进程,通过 Agent 进程,监视各种协议的 Client 与 Server 端的通信,用 LoadRunner 的一套 C 语言函数来录制脚本,所以只要 LoadRunner 支持的协议,就不会存在录制不到的,这是它与 Load test、WinRunner、Robot (Gui)录制脚本的一个很大区别(WinRunner 必须识别对象,才能录制到)。然后 LoadRunner 调用这些脚本向服务器端发出请求,接受服务器的响应,至于服务器内部如何处理,它并不关心。

7.2.2 创建虚拟用户

使用 LoadRunner 的 Virtual User Generator,可以很简便地创立起系统负载。该引擎能够生成虚拟用户,以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程(如下订单或机票预定),然后将其转化为测试脚本。利用虚拟用户,可以在 Windows、UNIX 或 Linux 机器上同时产生成千上万个用户访问。所以 LoadRunner 能极大地减少负载测试所需的硬件和人力资源。另外,LoadRunner 的 TurboLoad 专利技术能提供很高的适应性。TurboLoad 可以产生每天几十万名在线用户和数以百万计的点击数的负载。

用 Virtual User Generator 建立测试脚本后,可以对其进行参数化操作,这一操作能利用几套不同的实际发生数据来测试应用程序,从而反映出本系统的负载能力。以一个订单输入过程为例,参数化操作可将记录中的固定数据,如订单号和客户名称,由可变值来代替。在这些变量内随意输入可能的订单号和客户名,来匹配多个实际用户的操作行为。LoadRunner 通过它的 Data Wizard 来自动实现其测试数据的参数化。Data Wizard 直接连接数据库服务器,从中可以获取所需的数据(如订单号和用户名)并直接将其输入到测试脚本。这样避免了人工处理数据的需要,Data Wizard 节省了大量的时间。

为了进一步确定 Virtual user 能够模拟真实用户,可利用 LoadRunner 控制某些行为特性。例如,只需要单击一下鼠标,就能轻易控制交易的数量,交易频率,用户的思考时间和连接速度等。

7.2.3 创建真实的负载

Virtual users 建立起后,需要设定负载方案,业务流程组合和虚拟用户数量。用 LoadRunner 的 Controller,能很快组织起多用户的测试方案。Controller 的 Rendezvous 功

能提供一个互动的环境,在其中既能建立起持续且循环的负载,又能管理和驱动负载测试方案。而且,可以利用它的日程计划服务来定义用户在什么时候访问系统以产生负载。这样,就能将测试过程自动化。同样还可以用 Controller 来限定负载方案,在这个方案中所有的用户同时执行一个动作(如登录到一个库存应用程序)来模拟峰值负载的情况。另外,还能监测系统架构中各个组件的性能(包括服务器、数据库、网络设备等)来帮助客户决定系统的配置。

LoadRunner 通过它的 AutoLoad 技术,提供更多的测试灵活性。使用 AutoLoad,可以根据目前的用户人数事先设定测试目标,优化测试流程。例如,目标可以是确定应用系统承受的每秒点击数或每秒的交易量。

7.2.4 实时监测器

LoadRunner 内含集成的实时监测器,在负载测试过程的任何时候,都可以观察到应用系统的运行性能。这些性能监测器为实时显示交易性能数据(如响应时间)和其他系统组件包括 application server,web server、网络设备和数据库等的实时性能。这样,就可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能,从而更快地发现问题。

再者,利用 LoadRunner 的 ContentCheck TM,可以判断负载下的应用程序功能是否正常。ContentCheck 在 Virtual users 运行时,检测应用程序的网络数据包内容,从中确定是否有错误内容传送出去。它的实时浏览器有助于从终端用户角度观察程序性能状况。

7.2.5 分析结果

一旦测试完毕后,LoadRunner 收集汇总所有的测试数据,并提供高级的分析和报告工具,以便迅速查找到性能问题并追溯原由。使用 LoadRunner 的 Web 交易细节监测器,可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。例如,这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或是第三方的数据组件造成应用系统运行速度减慢。另外,Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间,便于确认问题,定位查找真正出错的组件。例如,可以将网络延时进行分解,以判断 DNS 解析时间,连接服务器或 SSL 认证所花费的时间。通过使用 LoadRunner 的分析工具,能很快地查找到出错的位置和原因并作出相应的调整。

7.2.6 重复测试

负载测试是一个重复过程,每次处理完一个出错情况,都需要对应用程序在相同的方案下,再进行一次负载测试。以此检验所做的修正是否改善了运行性能。

7.2.7 其他特性

利用 LoadRunner,可以很方便地了解系统的性能。它的 Controller 允许重复执行与出错修改前相同的测试方案。它的基于 HTML 的报告提供一个比较性能结果所需的基准,以此衡量在一段时间内,有多大程度的改进并确保应用成功。由于这些报告是基于 HTML 的文本,可以将其公布于公司的内部网上,便于随时查阅。

所有 Mercury Interactive 的产品和服务都是集成设计的,能完全相容地一起运作。由于它们具有相同的核心技术,来自于 LoadRunner 和 ActiveTest TM 的测试脚本,在

Mercury Interactive 的负载测试服务项目中，可以被重复用于性能监测。借助 Mercury Interactive 的监测功能——Topaz TM 和 ActiveWatch TM，测试脚本可重复使用从而平衡投资收益。更重要的是，能为测试的前期部署和生产系统的监测提供一个完整的应用性能管理解决方案。

1. Enterprise Java Beans 的测试

LoadRunner 完全支持 EJB 的负载测试。这些基于 Java 的组件运行在应用服务器上，提供广泛的应用服务。通过测试这些组件，可以在应用程序开发的早期就确认并解决可能产生的问题。

2. 支持无线应用协议

随着无线设备数量和种类的增多，测试计划需要同时满足传统的基于浏览器的用户和无线互联网设备，如手机和 PDA。LoadRunner 支持两项最广泛使用的协议：WAP 和 I-mode。此外，通过负载测试系统整体架构，使用 LoadRunner 可以只通过记录一次脚本，就可完全检测上述这些无线互联网系统。

3. 支持 Media Stream 应用

LoadRunner 还能支持 Media Stream 应用。为了保证终端用户得到良好的操作体验和高质量 Media Stream，需要检测 Media Stream 应用程序。使用 LoadRunner，可以记录和重放任何流行的多媒体数据流格式来诊断系统的性能问题，查找原由，分析数据的质量。

4. 完整的企业应用环境的支持

LoadRunner 支持广泛的协议，可以测试各种 IT 基础架构。

7.3 使用 LoadRunner 进行负载/压力测试——以 Web 应用为例

LoadRunner 包含很多组件，其中最常用的有 Visual User Generator（以下简称 VuGen）、Controller 和 Analysis。

使用 LoadRunner 进行测试的过程可以用图 7.1 表示。

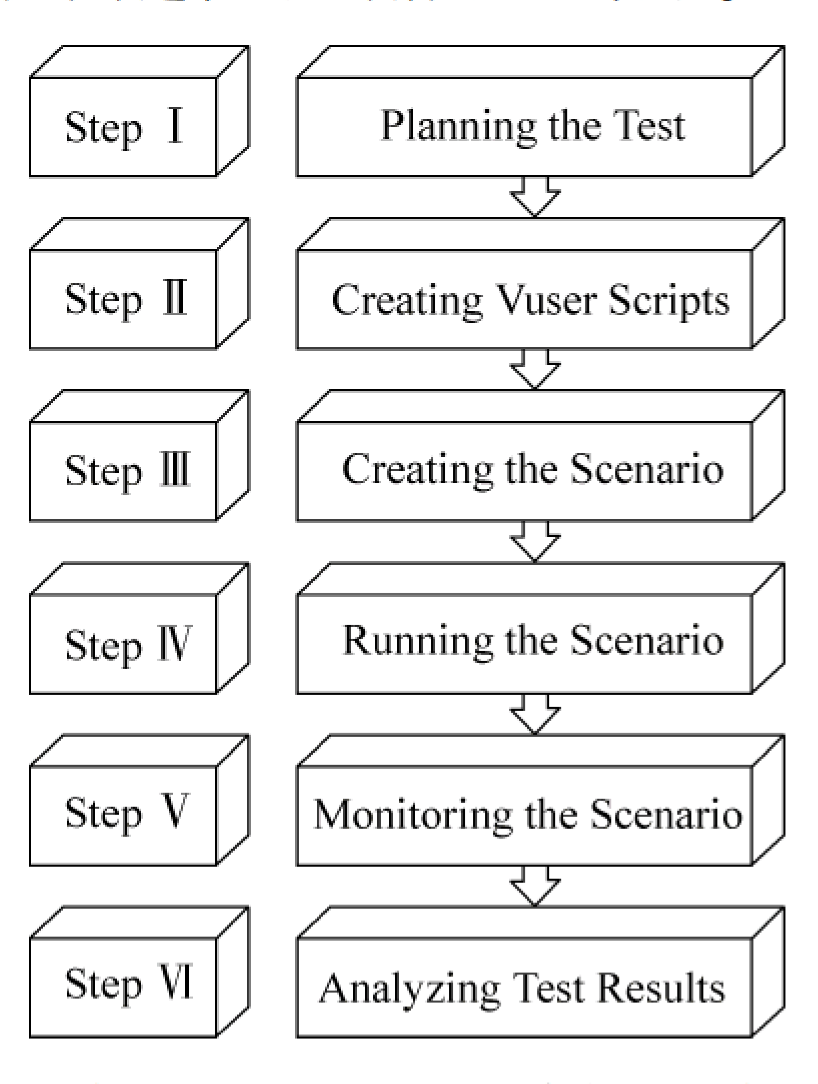


图 7.1　LoadRunner 测试过程图

下面按照图 7.1 的步骤来简单说明使用 LoadRunner 的测试过程。

7.3.1 制定负载测试计划

在任何类型的测试中，测试计划都是必要的步骤。测试计划是进行成功的负载测试的关键。任何类型的测试的第一步都是制定比较详细的测试计划。一个比较好的测试计划能够保证 LoadRunner 成功完成负载测试的目标。

制定负载测试计划一般情况下需要 3 个步骤，可以用图 7.2 表示。

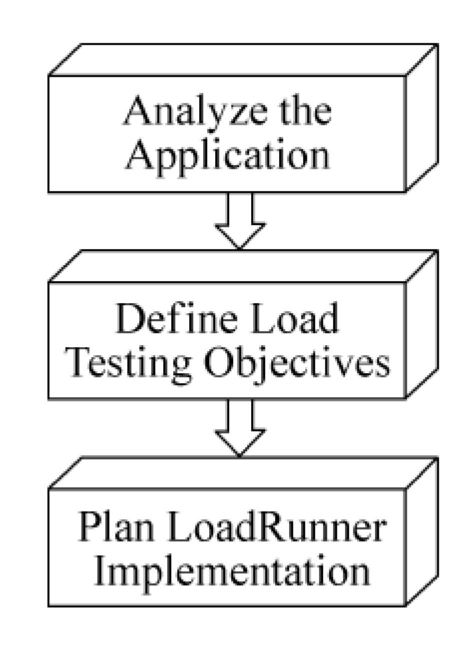


图 7.2 制定负载测试计划步骤图

1. 分析应用程序（Analyze the Application）

制定负载测试计划的第一步是分析应用程序。对系统的软硬件以及配置情况非常熟悉，才能保证使用 LoadRunner 创建的测试环境真实的反映实际运行的环境。

（1）确定系统的组成

画出系统的组成图。组成图主要包括系统中所有的组件，以及相互之间是如何通信的。图 7.3 为一个系统组成图的例子。

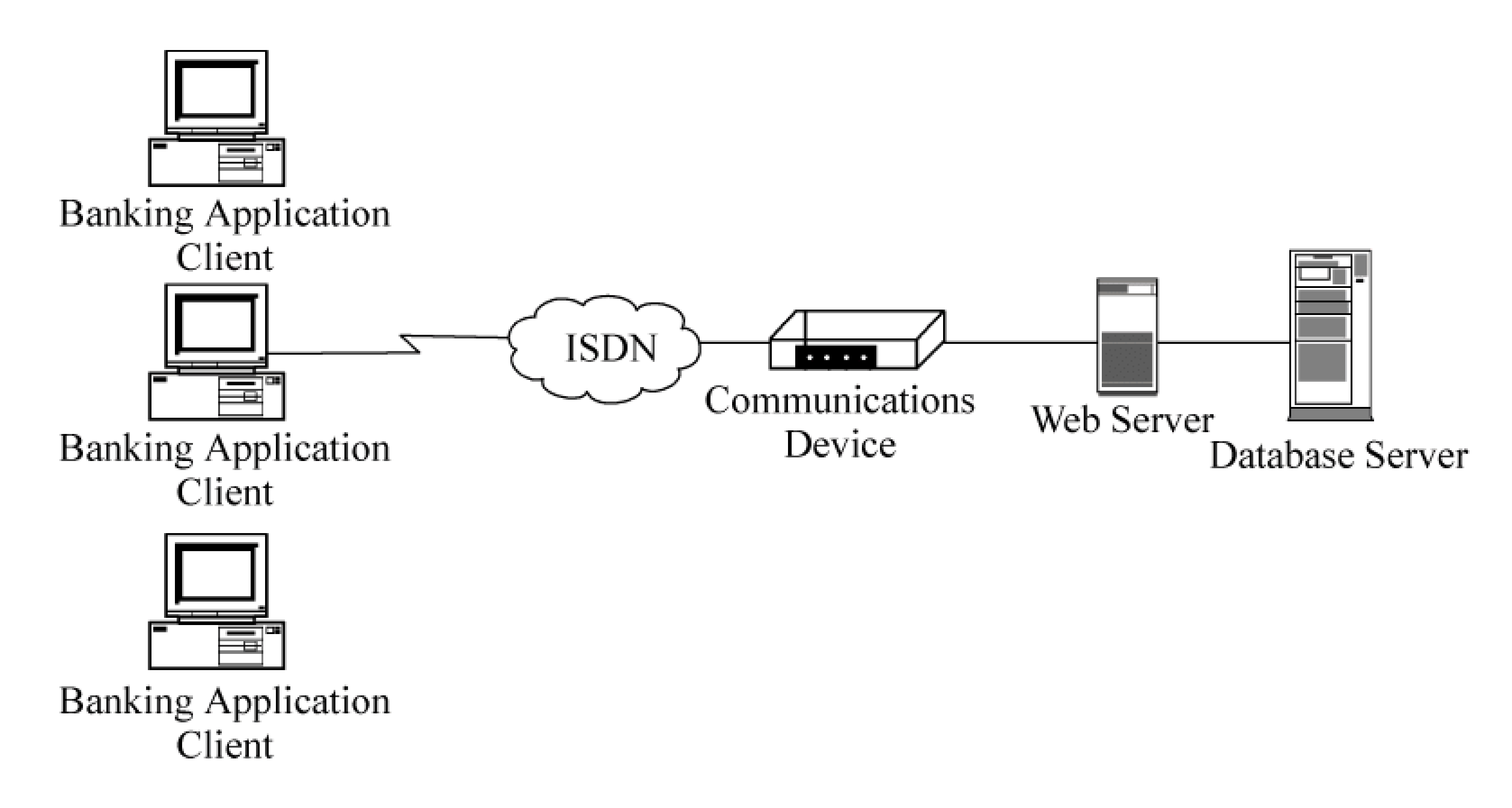


图 7.3 系统组成图

（2）描述系统配置

画出系统组成图后，试着回答以下问题，对组成图进行完善。

- 预计有多少用户会连接到系统；
- 客户机的配置情况（硬件、内存、操作系统、软件工具等）；
- 服务器使用什么类型的数据库以及服务器的配置情况；
- 客户机和服务器之间如何通信；
- 还有什么组件会影响 Response Time 指标（比如 Modem 等）；
- 通信装置（网卡、路由器等）的吞吐量是多少？每个通信装置能够处理多少并发用户。

（3）分析最普遍的使用方法

了解该系统最常用的功能，确定哪些功能需要优先测试、什么角色使用该系统以及每个角色会有多少人、每个角色的地理分布情况等，从而预测负载的最高峰出现的情况。

2. 确定测试目标(Defining Testing Objectives)

这里借用一图表来说明如何确定测试目标,如表 7.1 所示。

表 7.1 如何确定测试目标

Objective	Answers the Question
Measuring end-user response time	How long does it take to complete a business process?
Defining optimal hardware configuration	Which hardware configuration provides the best performance?
Checking reliability	How hard or long can the system work without errors or failures?
Checking hardware or software upgrades	How does the upgrade affect performance or reliability?
Evaluating new products	Which server hardware or software should you choose?
Measuring system capacity	How much load can the system handle without significant performance degradation?
Identifying bottlenecks	Which element is slowing down response time?

在这里还要确定何时开始负载测试,在不同的阶段进行什么内容的负载测试。这里用表 7.2 来说明。

表 7.2 不同阶段的测试内容

Planning and Design	Development	Deployment	Production	Evolution
Evaluate new products	Measure response time	Check reliability	Measure response time	Check HW or SW upgrades
Measure response time	Check optimal hardware configuration	Measure response time	Identify bottlenecks	Measure system capacity
	Check HW or SW upgrades	Measure system capacity		
	Check reliability			

3. 计划如何执行 LoadRunner

确定要使用 LoadRunner 度量哪些性能参数,根据测量结果计算哪些参数,从而可以确定 Vusers(虚拟用户)的活动,最终可以确定哪些是系统的瓶颈等。在这里还要选择测试环境,测试机器的配置情况等。

7.3.2 开发负载测试脚本

LoadRunner 使用虚拟用户的活动模拟真实用户来操作 Web 应用程序,而虚拟用户的活动就包含在测试脚本中,所以测试脚本对于测试来说是非常重要的。开发测试脚本要使用 VuGen 组件,测试脚本要完成如下内容。

- 每一个虚拟用户的活动
- 定义结合点
- 定义事务

开发测试脚本需要几个步骤,如图 7.4 所示。

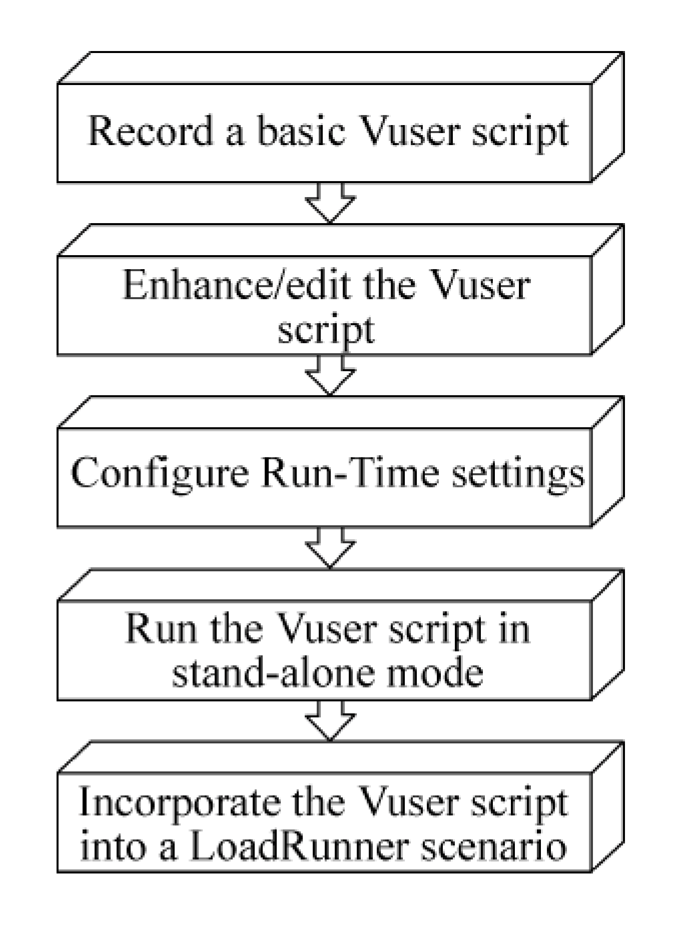


图 7.4 开发测试脚本过程步骤图

1. 录制测试脚本

启动 Visual User Generator,如图 7.5 所示,执行【File】→【New】命令新建一个用户脚本。



图 7.5 Virtual User Generator 窗口

在弹出的菜单中选择合适的通信协议。

一般情况下,B/S 系统选择 Web(Http/Html)。C/S 系统,根据 C/S 结构所用到的后台数据库来选择不同的协议,如果后台数据库是 Sybase,则采用 sybaseCTlib 协议,如果是 SQL server,则使用 MS SQL Server 的协议,至于 Oracle 数据库系统,使用 Oracle 2-tier 协议。没有数据库的 C/S(FTP,SMTP),可以选择 Windows Sockets 协议。其他的 ERP,EJB(需要 ejbdetector.jar),选择相应的协议即可。

这里需要测试的是 Web 应用,所以需要选择 Web(HTTP/HTML)协议,如图 7.6 所示。

单击【OK】按钮,进入主窗体,如图 7.7 所示。在菜单栏中执行【Vuser】→【Start Recording】命令或者在工具栏中单击 Start Record 按钮,都可以启动录制脚本的命令,打开录制窗口,如图 7.8 所示。

在 URL 中添入要测试的 Web 站点地址,这里以 MercuryWebTours 应用为例子来进行录制。

选择要把录制的脚本放到哪一个部分,VuGen 中的脚本分为 3 部分:vuser_init、vuser_end 和 Action。默认情况下是“Action”。Action 可以通过单击【New】按钮,新建 ActionX,将其分成多个部分。而 vuser_init 和 vuser_end 都只能有一个,不能再分。

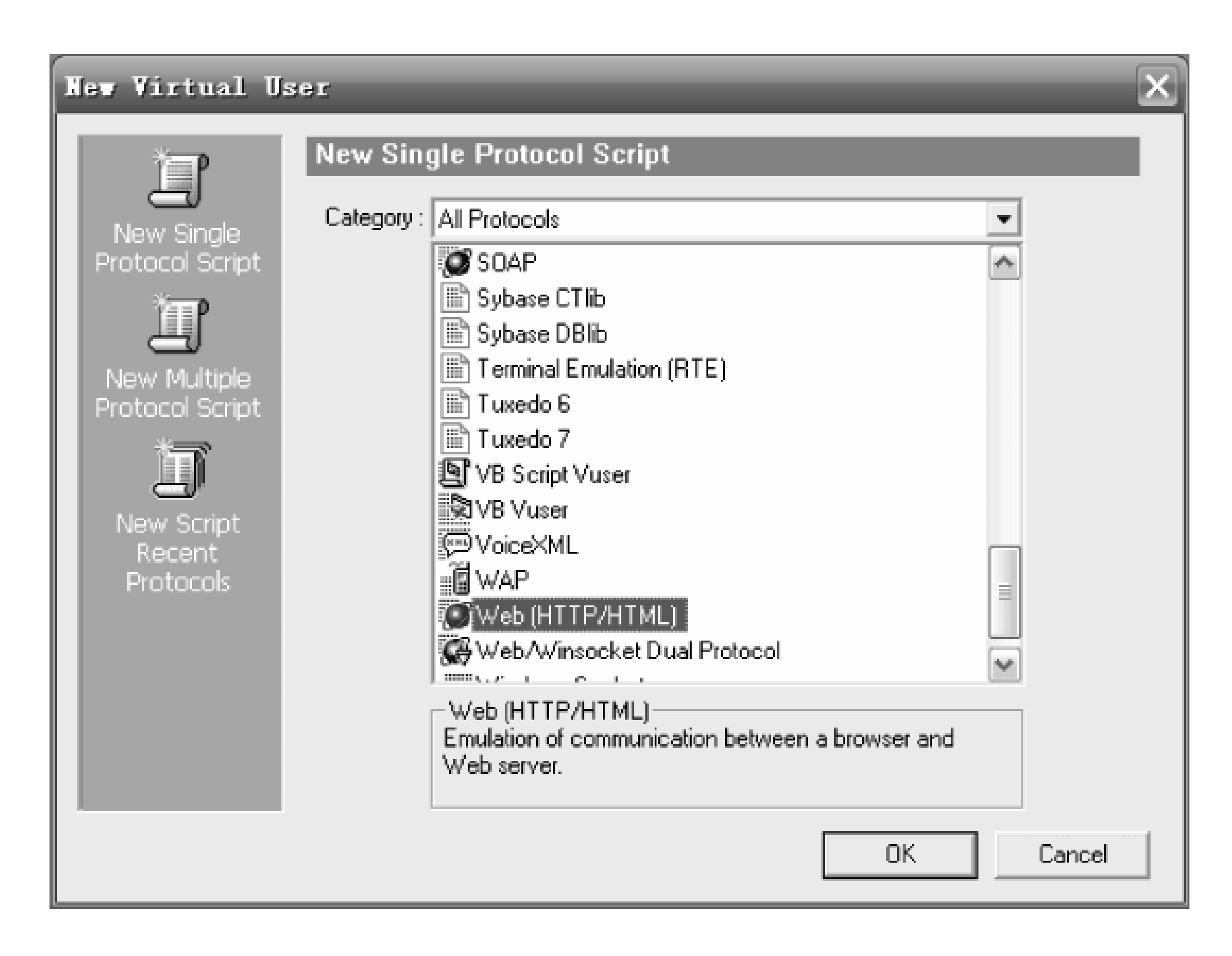


图 7.6　New Virtual User 窗口

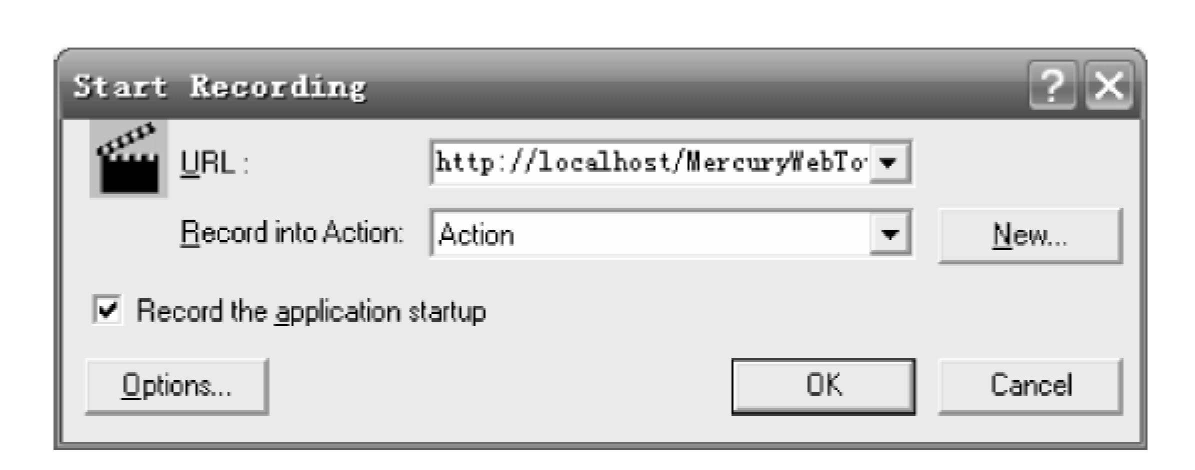


图 7.7　Virtual User Generator 主窗体

图 7.8　Start Recording 窗体

在录制需要登录的系统时，把登录部分放到 vuser_init 中，把登录后的操作部分放到 Action 中，把注销关闭登录部分放到 vuser_end 中。如果需要在登录操作设集合点，那么登录操作也要放到 Action 中，因为 vuser_init 中不能添加集合点。

注意：*在重复执行测试脚本时，vuser_init 和 vuser_end 中的内容只会执行一次，重复执行的只是 Action 中的部分。*

“Record the application startup”默认情况下是选中的，表示一旦应用程序启动，VuGen 就会开始录制脚本；如果没有选中，应用程序启动后 VuGen 出现如图 7.9 所示对话框，并且不会开始录制脚本，用户操作应用程序到需要录制的地方，单击“Record”按钮，VuGen 才开始录制。



图 7.9　Recording Suspended 对话框

在录制窗口点【Options】按钮，进入录制的设置窗体，这里一般情况下不需要改动。

Recording 标签页：默认情况下选择【HTML-based script】，如图 7.10 所示，说明脚本中采用 HTML 页面的形式来表示，这种方式的 Script 脚本容易维护，容易理解。【URL-based script】项说明脚本中的表示采用基于 URL 的方式，这种方式看上去较乱。

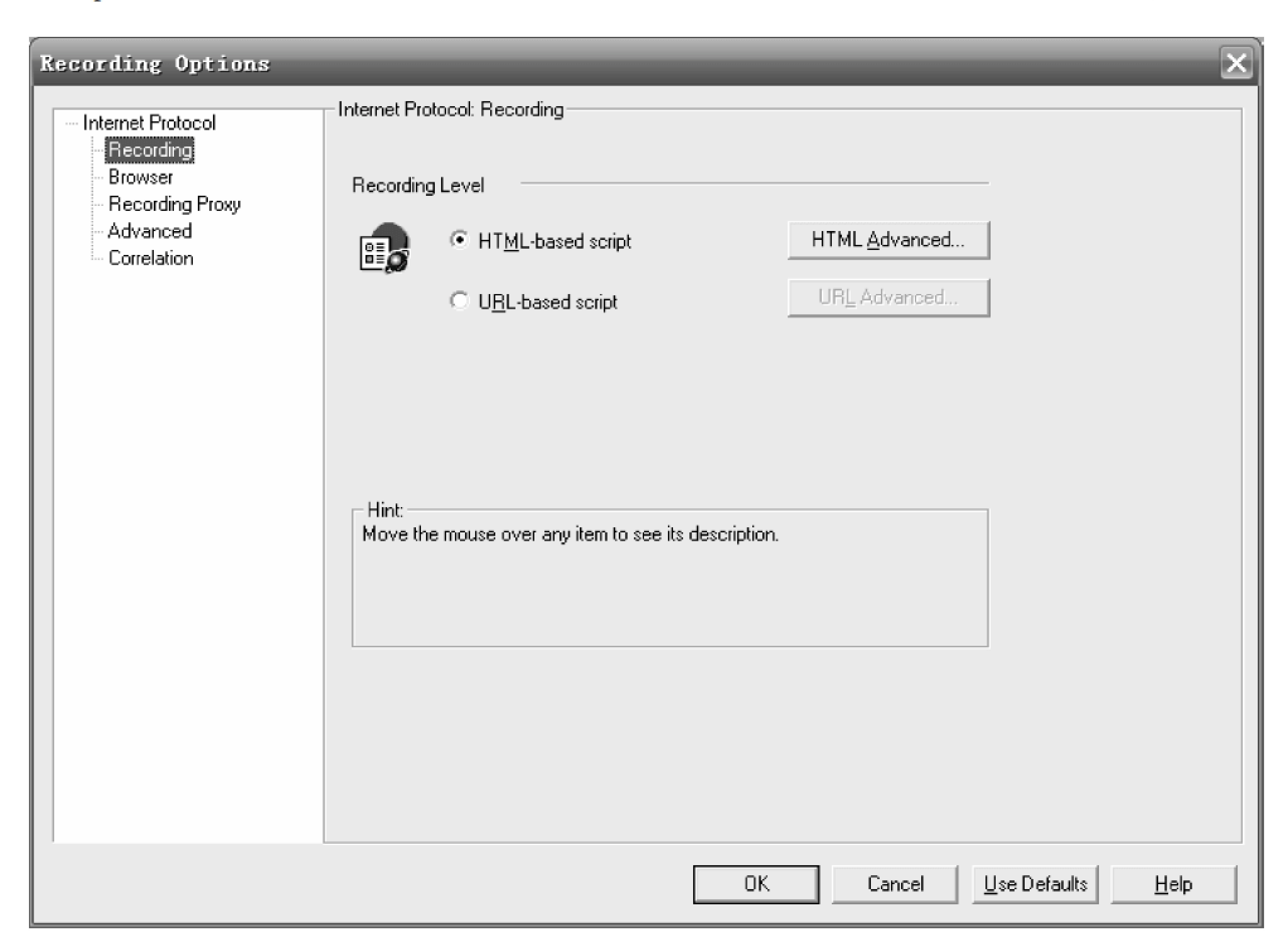


图 7.10　Recording Options 窗体

选择哪种方式录制，有以下参考原则。

(1) 基于浏览器的应用程序推荐使用 HTML-based script；

(2) 不是基于浏览器的应用程序推荐使用 URL-based script；

(3) 如果基于浏览器的应用程序中包含了 JavaScript 并且该脚本向服务器发出了请求，比如 DataGrid 的分页按钮等，也要使用 URL-based 方式录制；

(4) 基于浏览器的应用程序中使用了 HTTPS 安全协议，使用 URL-based 方式录制。

Advanced 标签页：取默认情况即可。

Correlation 标签页：这里的内容比较重要，需要定制，主要是为了在录制过程中设置自动关联，根据自己的需求，选择适当的设置，然后单击【OK】按钮，VuGen 开始录制脚本。

录制过程中，在屏幕上会出现如下一个工具条。



下面简单介绍一下各个按钮的功能。

●：开始录制　　　　▮▮：暂停录制

■：结束录制　　　　▶：运行测试脚本

：开始编译　　　　Action：创建一个新的 Action

：插入 Text 检查点　　　　：插入事务的“起始点”

：插入事务的“结束点”　　　　：插入“集合点”

：插入注释　　　　：改变录制的 options 设置

按照计划完成录制的过程。

录制完成后，按下 ■ 按钮，结束录制。VuGen 自动生成用户脚本，退出录制过程。录制的用户脚本参考图 7.11。

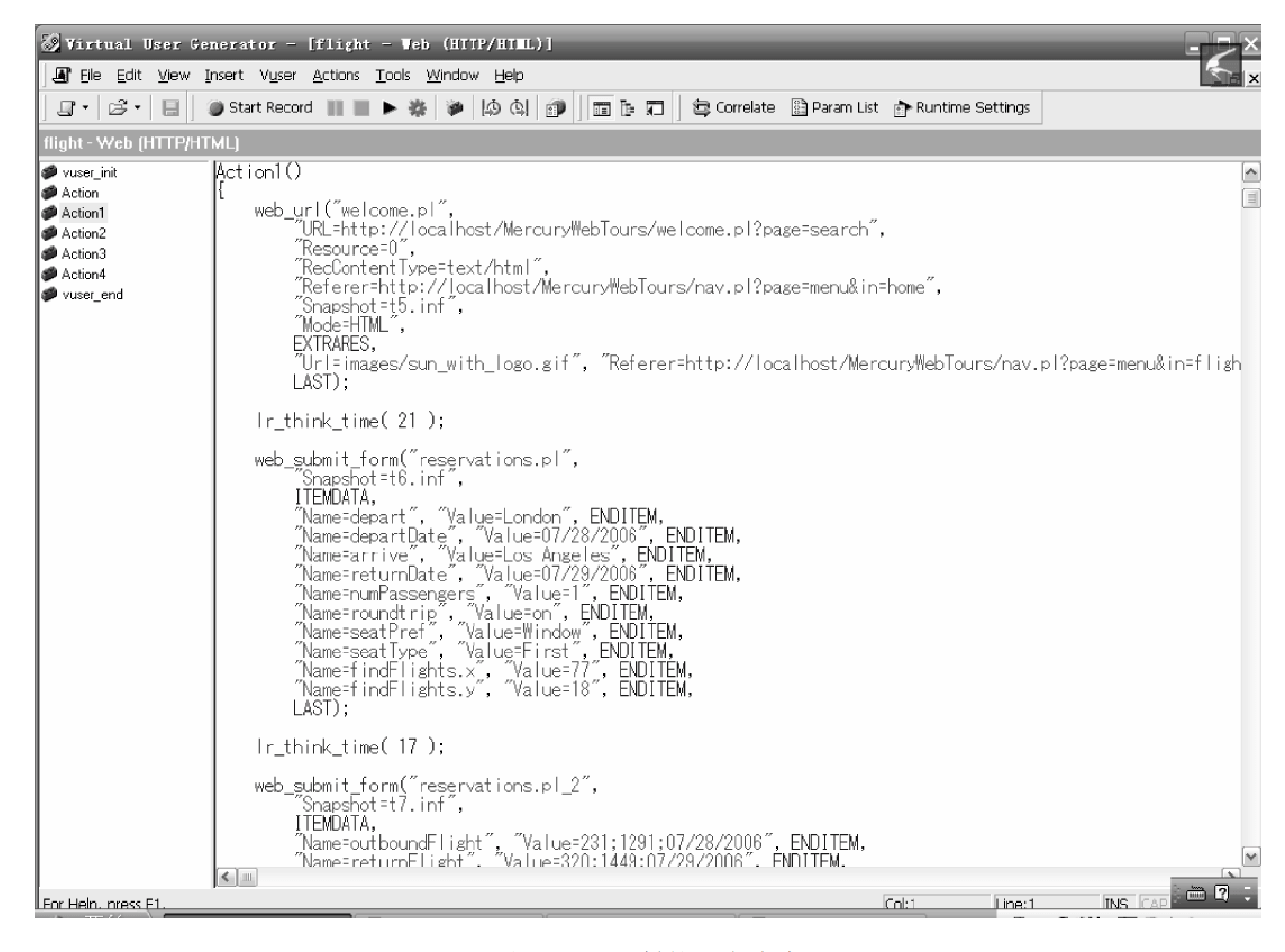


图 7.11　录制的用户脚本

测试脚本录制/分配遵循以下一些原则。

(1) 脚本越小越好。就像写 code 一样的，不要太长，尽量做到一个功能(Transaction)一个脚本。如果那些功能是连续有序的，必须先做上一个，才能做下一个，那就只好放在一起了。

(2) 选择使用频率最高的。有些人喜欢在 LoadRunner 中测试几乎所有的功能，其实这样不合适，把最常用的、使用频率最高的拿出来测试。但是也要结合用户实际使用情况，一般在一个系统中是多个用户使用多个功能，某些功能使用的频率更大一些，在录制脚本之前就要设计好，哪个脚本会跑几个用户，一共需要多少个脚本，能满足性能测试的需求。

(3) 选择所需要的进行录制。对于 Web 的程序，对于所关注的内容没什么影响的操作，可以不录制，可以使用暂停，这需要对被测功能有一个清楚的认识和了解，要能把握住哪

些地方是对整个过程没有影响的，比如一些查询，通常，选择条件的页面都可以不录制，但对于一些页面有可能要传递参数，就需要录制了，如何确定哪些点可以不录制，一是可以找开发人员了解清楚程序设计的结构，再就是靠自己的经验。

2. 完善测试脚本

当录制完一个基本的用户脚本后，在正式使用前还需要完善测试脚本，增强脚本的灵活性。一般情况下，可以通过以下方法来完善测试脚本。

(1) 插入事务

事务(Transaction)：为了衡量服务器的性能，需要定义事务。比如，在脚本中有一个数据查询操作，为了衡量服务器执行查询操作的性能，把这个操作定义为一个事务，这样在运行测试脚本时，LoadRunner 运行到该事务的开始点就开始计时，直到运行到该事务的结束点，计时结束。这个事务的运行时间将会反映在测试结果中。

插入事务操作可以在录制过程中进行，也可以在录制结束后进行。LoadRunner 允许在脚本中插入不限数量的事务。

具体的操作方法如下：在需要定义事务的操作前面，通过执行【Insert】→【Start Transaction】命令或者在工具栏单击 按钮，会出现如图 7.12 所示对话框。

输入该事务的名称，事务的名称最好要有意义，能够清楚的说明该事务完成的动作。

插入事务的“开始点”后，应该在需要定义事务的操作后面插入事务的“结束点”。同样可以通过执行【Insert】→【End Transaction】命令或者在工具栏上单击 按钮，会出现如图 7.13 所示对话框。

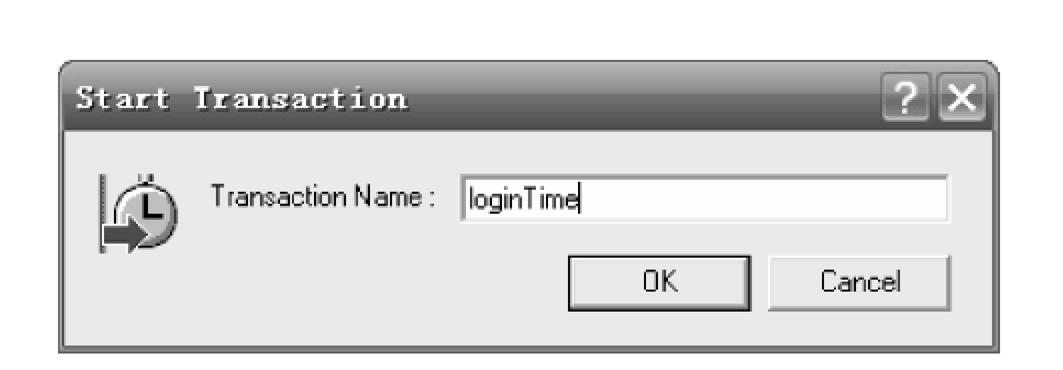


图 7.12　Start Transaction 窗体

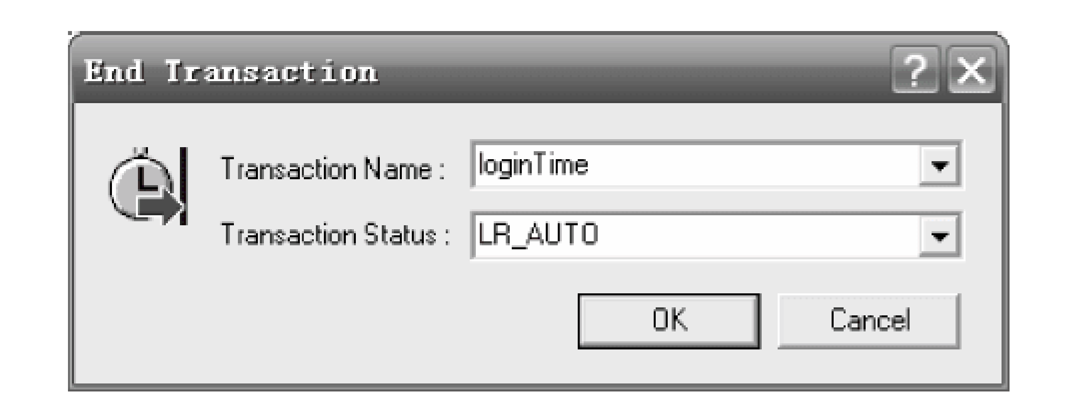


图 7.13　End Transaction 窗体

默认情况下，事务的名称列出最近的一个事务名称，事务的状态是 LR_AUTO。一般情况下，事务名称和状态都不需要修改，除非在手工编写代码时，有可能需要手动设置事务的状态。

脚本中事务的代码如下：

```
lr_start_transaction("loginTime");
```

此处为事务操作语句

```
lr_end_transaction("loginTime",LR_AUTO);
```

(2) 插入集合点

插入集合点是为了衡量在加重负载的情况下服务器的性能情况。在测试计划中，可能会要求系统能够承受 1000 人同时提交数据，在 LoadRunner 中，可以在提交数据操作前面加入集合点，这样当虚拟用户运行到提交数据的集合点时，LoadRunner 就会检查同时有多少用户运行到集合点，如果不到 1000 人，LoadRunner 就会命令已经到集合点的用户在此等待，当在集合点等待的用户达到 1000 人时，LoadRunner 命令 1000 人同时去提交数据，从而

达到测试计划中的需求。

注意：集合点经常和事务结合起来使用。集合点只能插入到 Action 部分，vuser_init 和 vuser_end 中都不能插入集合点。

具体的操作方法如下：在需要插入集合点的前面，通过执行【Insert】→【Rendezvous】命令或者单击工具栏中 按钮，会出现如图 7.14 所示对话框。

输入该集合点的名称，这里为“flightBook”。集合点的名称最好能够清楚的说明该集合点完成的动作。

脚本中集合点的代码如下：

```
lr_rendezvous("flightBook");
```

(3) 插入注释

插入注释最好是在录制过程中。具体的操作方法如下：在需要插入注释的前面，通过执行【Insert】→【Comment】命令或者单击工具栏中 按钮，会出现如图 7.15 所示对话框。

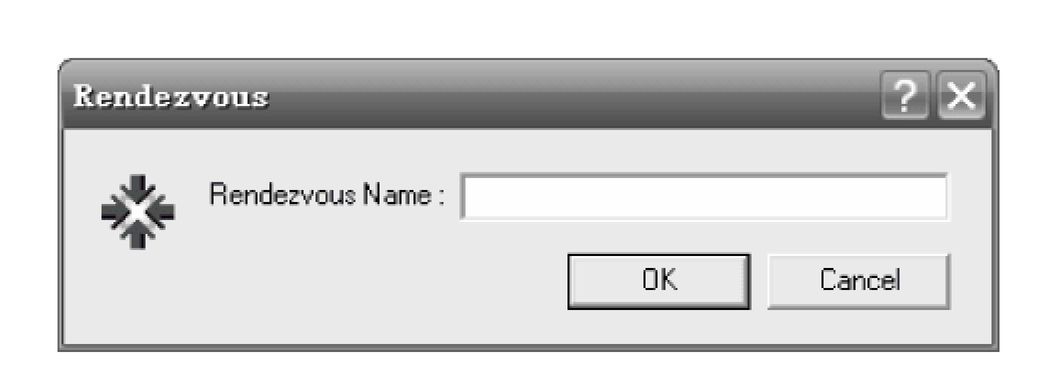



图 7.14　Rendezvous 对话框



图 7.15　Insert Comment 对话框

输入为代码添加的注释，就会在脚本中出现这样的注释代码：

```
/*
 * 注释语句
 */
```

(4) 参数化输入

如果用户在录制脚本过程中，填写提交了一些数据，比如要增加数据库记录，这些操作都被记录到了脚本中。当多个虚拟用户运行脚本时，都会提交相同的记录，这样不符合实际的运行情况，而且有可能引起冲突。为了更加真实的模拟实际环境，需要各种各样的输入。参数化输入是一种不错的方法。

用参数表示用户的脚本有以下两个优点。

① 可以使脚本的长度变短。

② 可以使用不同的数值来测试你的脚本。

参数化包含以下两项任务。

① 在脚本中用参数取代常量值。

② 设置参数的属性以及数据源。

参数化只能用于一个函数中的参量。不能用参数表示非函数参数的字符串。另外，不是所有的函数都可以参数化的。

参数化输入的讲解，采用一个例子的方式来进行。

```
web_submit_form("reservations.pl_3",
    "Snapshot=t8.inf",
    ITEMDATA,
    "Name=firstName", "Value=Joseph", ENDITEM,
    "Name=lastName", "Value=Marshall", ENDITEM,
    "Name=address1", "Value=234 Willow Drive", ENDITEM,
    "Name=address2", "Value=San Jose/CA/94085", ENDITEM,
    "Name=pass1", "Value=Joseph Marshall", ENDITEM,
    "Name=creditCard", "Value=", ENDITEM,
    "Name=expDate", "Value=", ENDITEM,
    "Name=saveCC", "Value=<OFF>", ENDITEM,
    "Name=buyFlights.x", "Value=80", ENDITEM,
    "Name=buyFlights.y", "Value=20", ENDITEM,
    LAST);
```

假如有以上的一个提交数据的窗体，如想参数化登录的用户名和密码，需要选中“Joseph”，然后单击鼠标右键。



选择“Replace with a parameter.”，出现如图 7.16 所示窗口，在窗口中填写参数名称，选择参数类型，确定其初始值。

Select or Create Parameter

Parameter name: NewParam

Parameter type: File

Original value: Joseph

OK　Cancel　Properties...

图 7.16　Select or Create Parameter 窗口

下面重点介绍一下参数类型。

- DateTime：在需要输入日期/时间的地方，可以用 DateTime 类型来替代。其属性设置很简单，选择一种格式即可，也可以定制格式。
- Group Name：在实际运行中，LoadRunner 使用该虚拟用户所在的 Vuser Group 来代替。但是在 VuGen 中运行时，Group Name 将会是 None。
- Load Generator Name：在实际运行中，LoadRunner 使用该虚拟用户所在 Load Generator 的机器名来代替。
- Iteration Number：在实际运行中，LoadRunner 使用该测试脚本当前循环的次数来代替。
- Random Number：随机数。在属性设置中可以设置产生随机数的范围。
- Unique Number：唯一的数。在属性设置中可以设置第一个数以及递增的数的大小。

注意：使用该参数类型必须注意可以接受的最大数。例如，某个文本框能接受的最大数为 99。当使用该参数类型时，设置第一个数为 1，递增的数为 1，但 100 个虚拟用户同时运行时，第 100 个虚拟用户输入的将是 100，这样脚本运行将会出错。这里说的递增意思是各个用户取第一个值的递增数，每个用户相邻的两次循环之间的差值为 1。如：假如起始数为 1，递增为 5，那么第一个用户第一次循环取值 1，第二次循环取值 2；第二个用户第一次循环取值为 6，第二次为 7；依次类推。

• Vuser ID：在实际运行中，LoadRunner 使用该虚拟用户的 ID 来代替，该 ID 是由 Controller 来控制的。但是在 VuGen 中运行时，Vuser ID 将会是－1。
• File：需要在属性设置中编辑文件，添加内容，也可以从现成的数据库中取数据，将会在下面进行介绍。
• User Defined Function：从用户开发的.dll 文件提取数据。
• Each Occurrence：运行时，每遇到一次该参数，就取一个新的值。
• Each iteration：运行时，在每一次循环中都取相同的值。
• Once：运行时，在每次循环中，该参数只取一次值。

这里要用数据库中的用户姓名来参数化用户姓名，参数名称为 firstName，选择 File 类型，初始值不变，如图 7.17 所示。

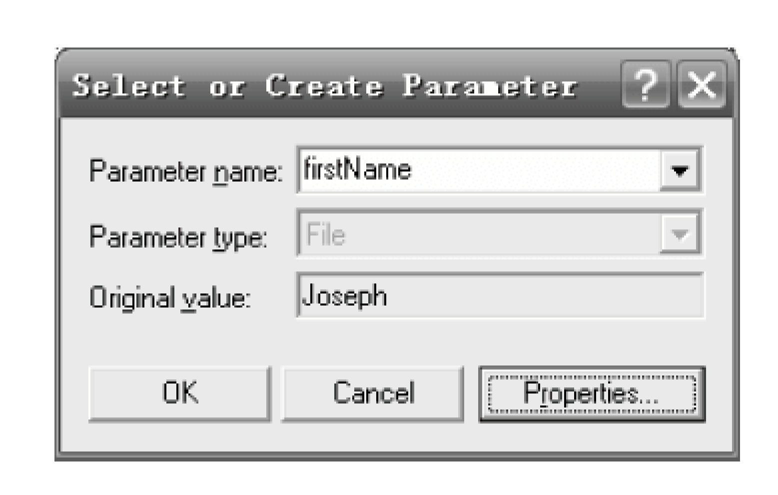


图 7.17　Select or Create Parameter 窗口

单击【Properties...】按钮，出现如图 7.18 所示窗口。

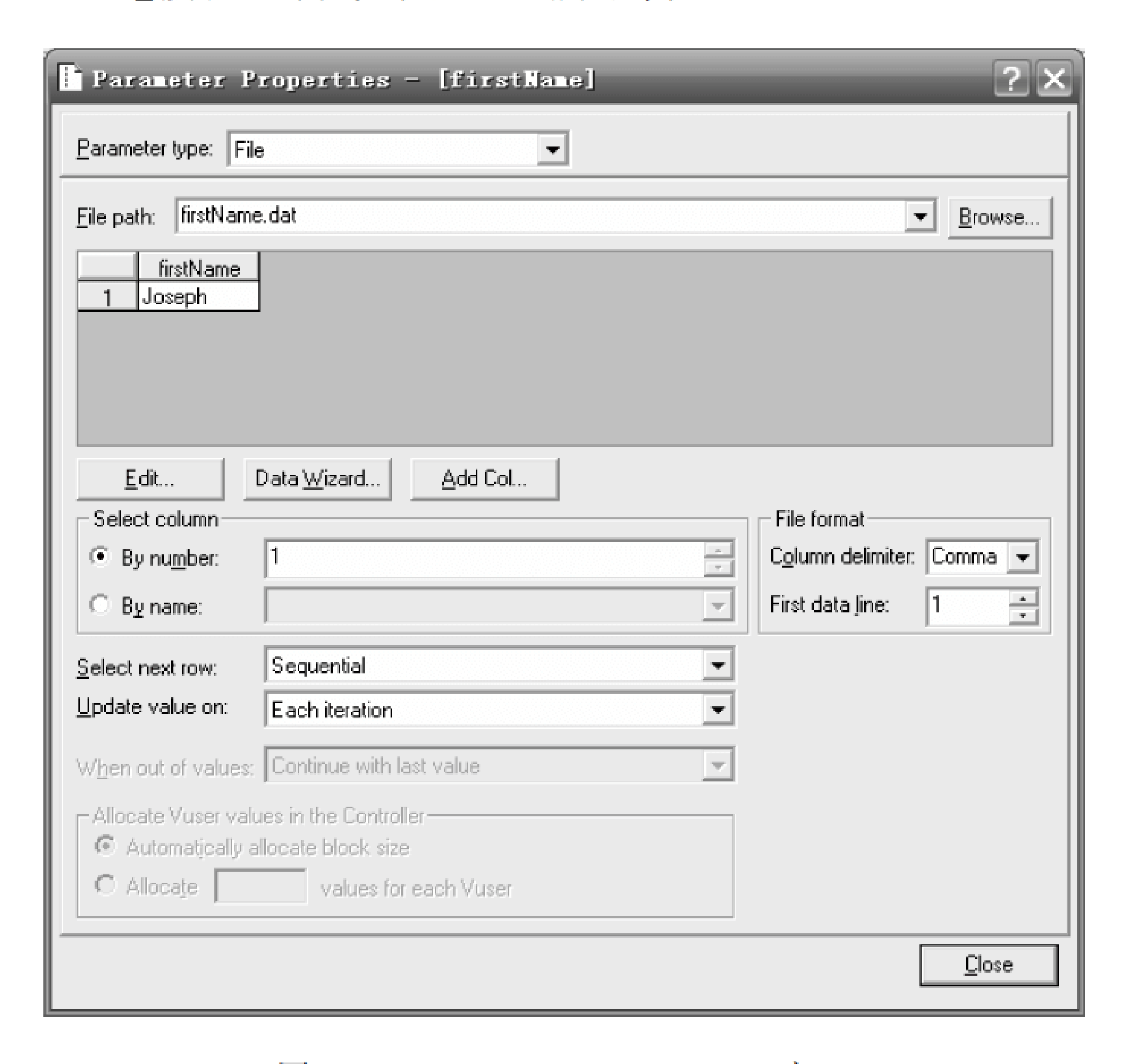


图 7.18　Parameter Properties 窗口

注意：这里用的是 LoadRunner 提供的实例，所以打开属性框就存在 firstName 的数据源，可以单击【Edit】按钮，进入一个记事本文件查看或添加所需要的数据，如图 7.19 所示。

【Select next row】有以下几种选择。

• Sequential：按照顺序一行行的读取。每一个虚拟用户都会按照相同的顺序读取。
• Random：在每次循环里随机的读取一个，但是在循环中一直保持不变。
• Unique：唯一的数。使用该类型必须注意数据表有足够多的数。比如 Controller 中设定 20 个虚拟用户进行 5 次循环，那么编号为 1 的虚拟用户取前 5 个数，编号为 2 的虚拟用户取 6～10 的数，依次类推，这样数据表中至少要有 100 个数据，否则

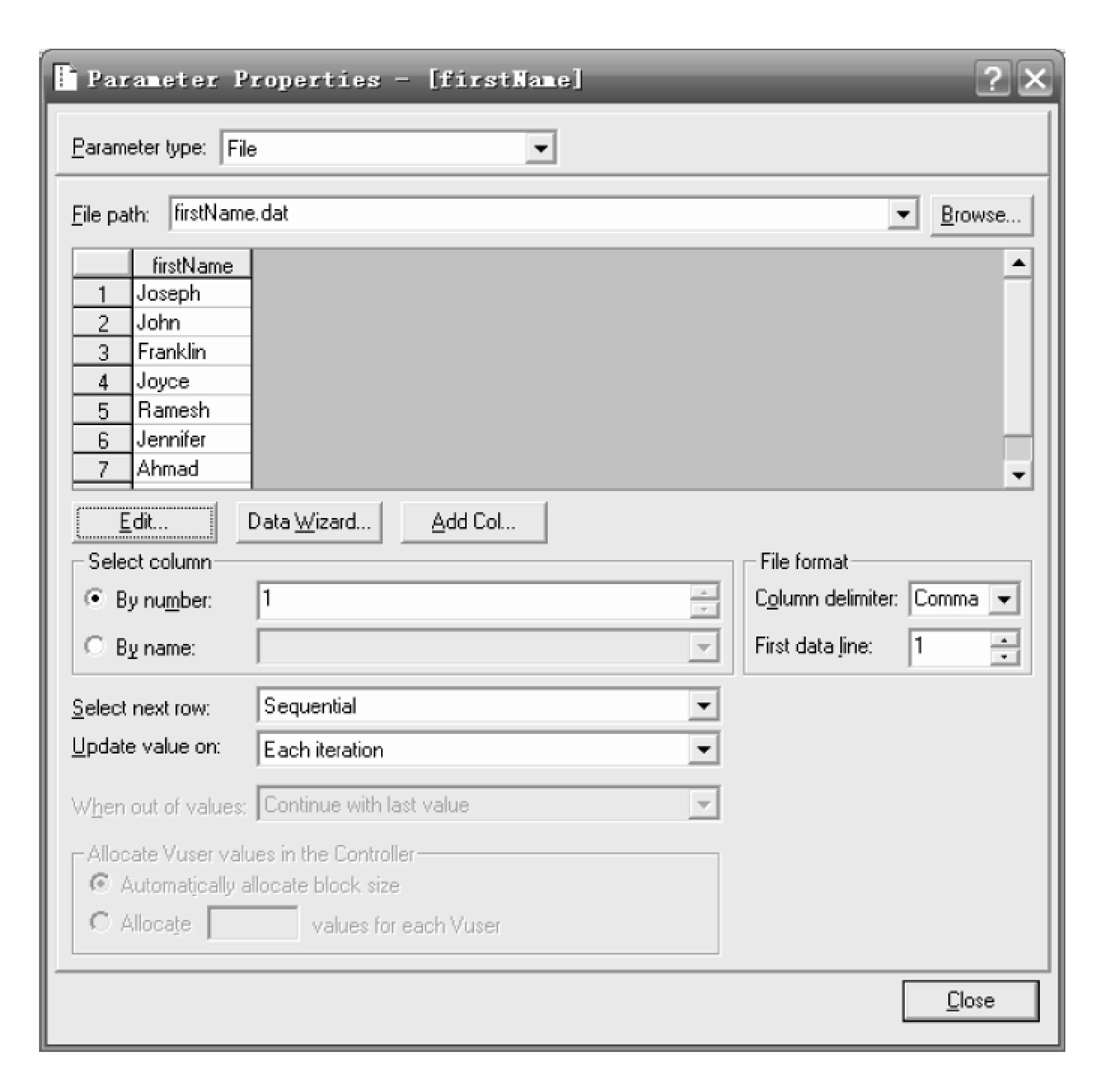


图 7.19　Parameter Properties 窗口

Controller 运行过程中会返回一个错误。

- Same Line As 某个参数(比如 Name)：和前面定义的参数 Name 取同行的记录。通常用在有关联性的数据上面，这里取值 Sequential 即可。Advance row each iteration 选中即可，表示每一次循环都往前走一行。

如果需要与数据库建立连接，从数据表中选择数据。单击【Data Wizard】按钮，选中【Specify SQL statement manually】选项(即使用 SQL 语言查询)，如图 7.20 所示。

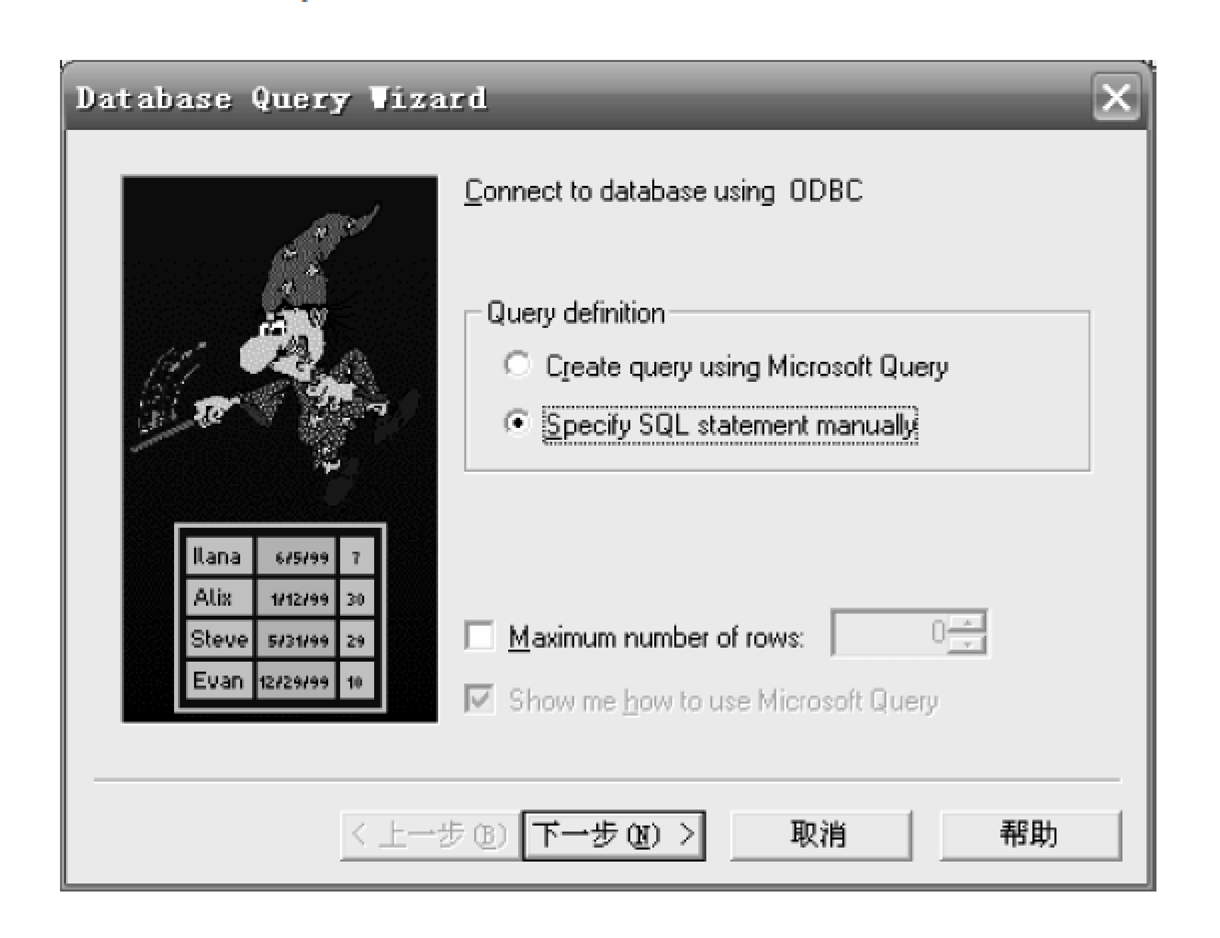


图 7.20　Database Query Wizard 窗口

单击【Create】按钮，建立好数据源连接后单击【确定】按钮返回，单击【Finish】按钮完成设置，如图 7.21 所示。

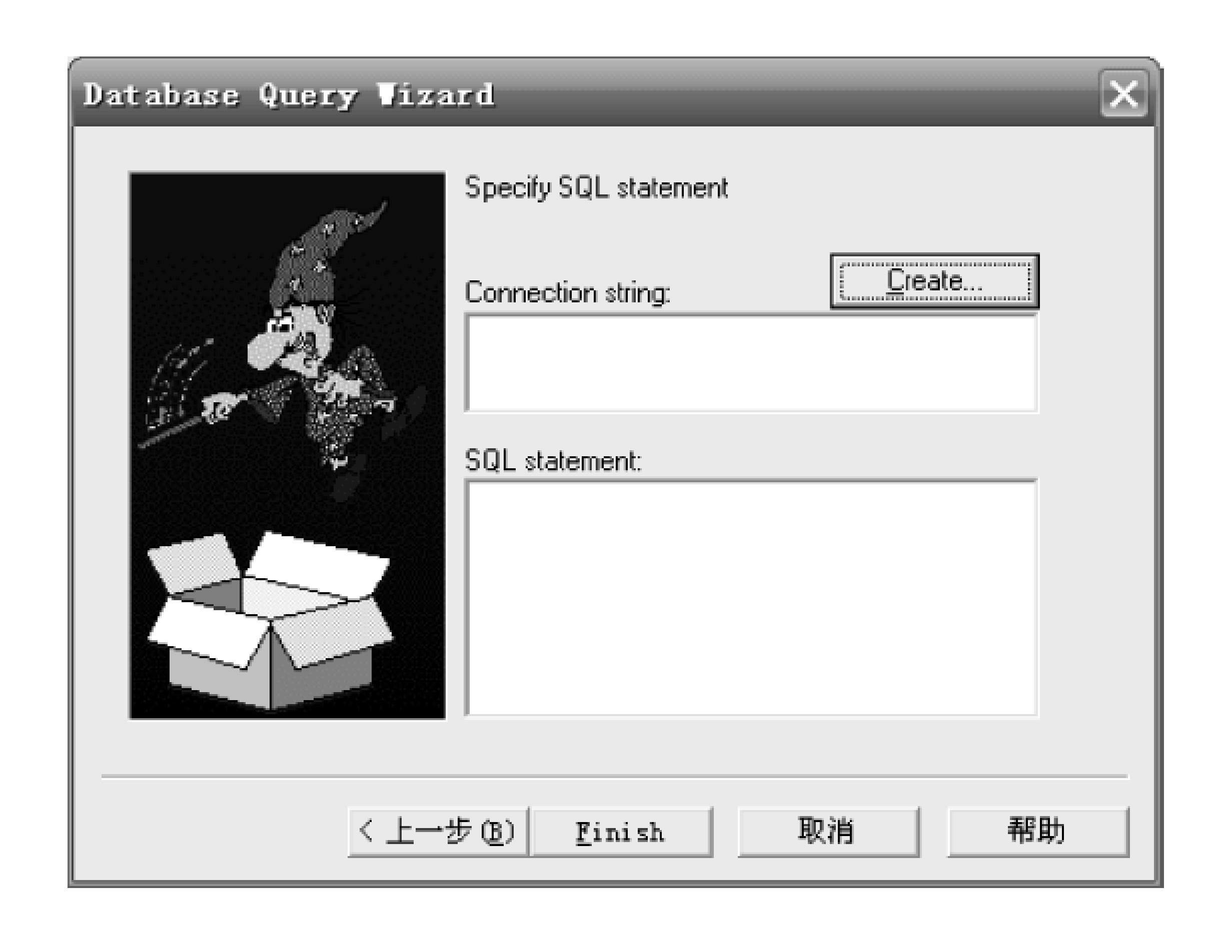


图 7.21　Database Query Wizard 窗口

(5) 插入函数

VuGen 中可以使用 C 语言中比较标准的函数和数据类型,语法和 C 语言相同。下面简单介绍一下比较常用的函数和数据类型。

① 控制脚本流程

```
if { } else { }
for{ }
while{ }
…
```

总之 C 语言的控制流程的语句这里都可以直接使用。

② 字符串函数

由于在 VuGen 脚本中使用最多的还是字符串,所以字符串函数在脚本中使用非常频繁。具体的语法请参考帮助说明。

strcmp 比较两个字符串

strcat 连接两个字符串

strcpy 复制字符串

……

注意:在 VuGen 中,以 char 声明的字符串是只读的,如果试图给 char 类型的字符串赋值的话,编译会通过,但在运行时会产生"Access Violation"的错误。解决这类问题,就是把字符串声明为字符数组,比如 char[100]。

③ 输出函数

输出函数在调试脚本时非常有用。

lr_output_message 输出一条消息

……

④ LoadRunner 提供的标准函数

lr_eval_string 函数功能是得到参数(参数化输入中)当前的值,例如:lr_output_message("temp=%s",lr_eval_string("{WCSParam2}"));

lr_save_string 函数功能是把一个字符串保存到参数中，例如：lr_save_string("439", "WCSParam3")；

(6) 插入 Text/Imag 检查点

在进行压力测试时，为了检查 Web 服务器返回的网页是否正确，VuGen 允许插入 Text/Imag 检查点，这些检查点验证网页上是否存在指定的 Text 或者 Imag，还可以在比较大的压力测试环境中测试被测的网站功能是否保持正确。

VuGen 在测试 Web 时，有两种视图方式：TreeView 和 Script View。这两种方式可以相互切换，在菜单中执行【View】→【Tree View】/【Script View】命令，或者单击工具栏中 (Script View)/ (Tree View)按钮。

前面见到的一直都是 Script View。在插入 Text/Imag 检查点时，用 TreeView 视图会比较方便些，在图 7.22 所示的 VuGen 主窗体中可以看到 TreeView 视图。

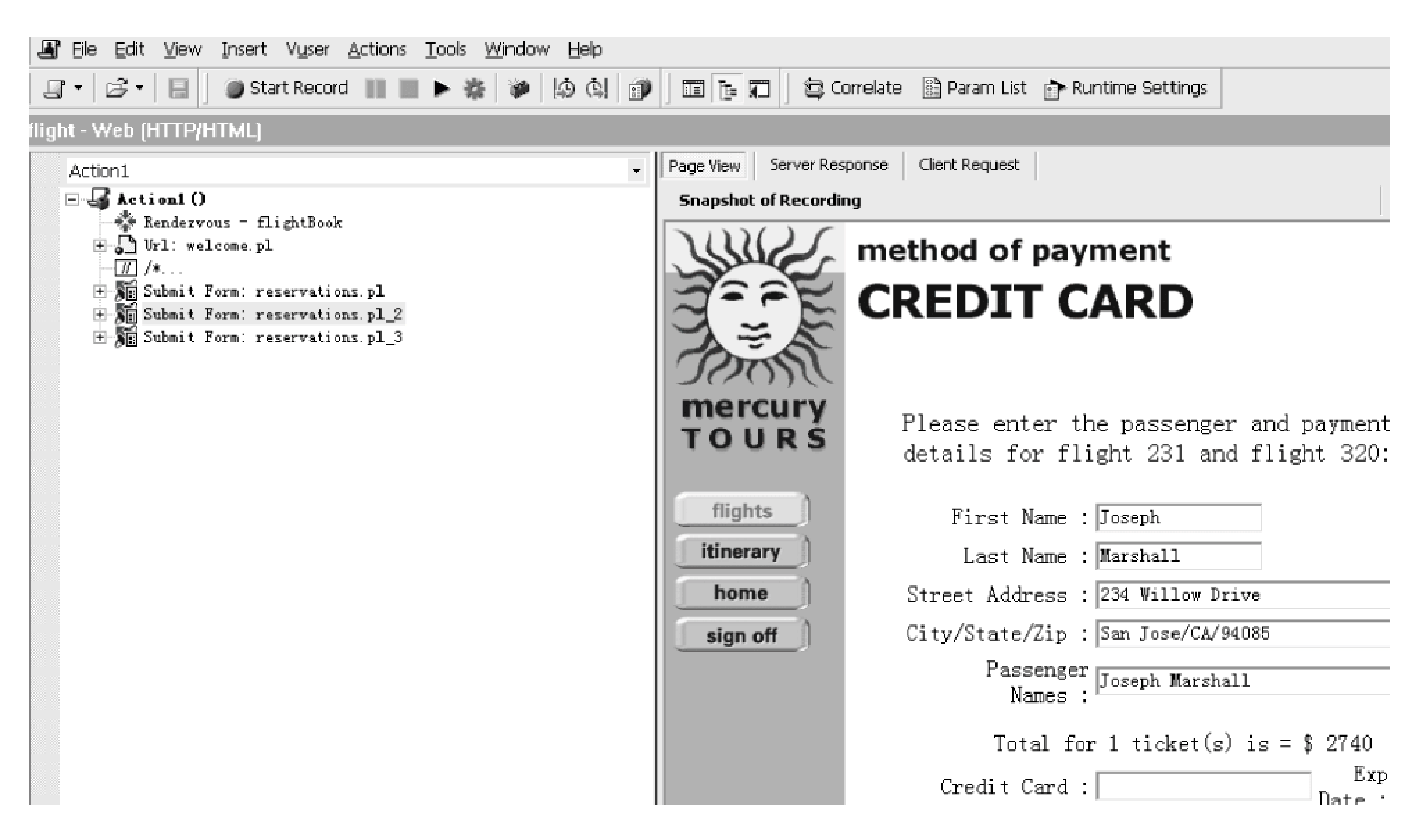


图 7.22 Virtual User Generator 主窗体

添加 Text/Imag 检查点，可以在录制过程中，也可以在录制完成后进行。最好能在录制过程中添加 Text/Imag 检查点。

先在树形菜单中选择需要插入检查点的一项，然后单击鼠标右键，选择将检查点插到该操作执行前还是该操作执行后。如果在该操作执行前，则选择【Insert Before】项，否则选择【Insert After】项，如图 7.23 所示。

然后弹出对话框，如图 7.24 所示，选择【Text Check】项(这里以 Text 检查点为例说明)。

单击【OK】按钮后，出现 Text Check Properties 对话框，如图 7.25 所示。

在【Search for】中输入需要在网页中搜索的字符或者字符串(字符串可以使用正则表达式)，如果需要定位搜索字符串左边和右边的字符或字符串，可以在【Right of】中输入搜索字符串右边的字符串，在【Left of】中输入搜索字符串左边的字符串。

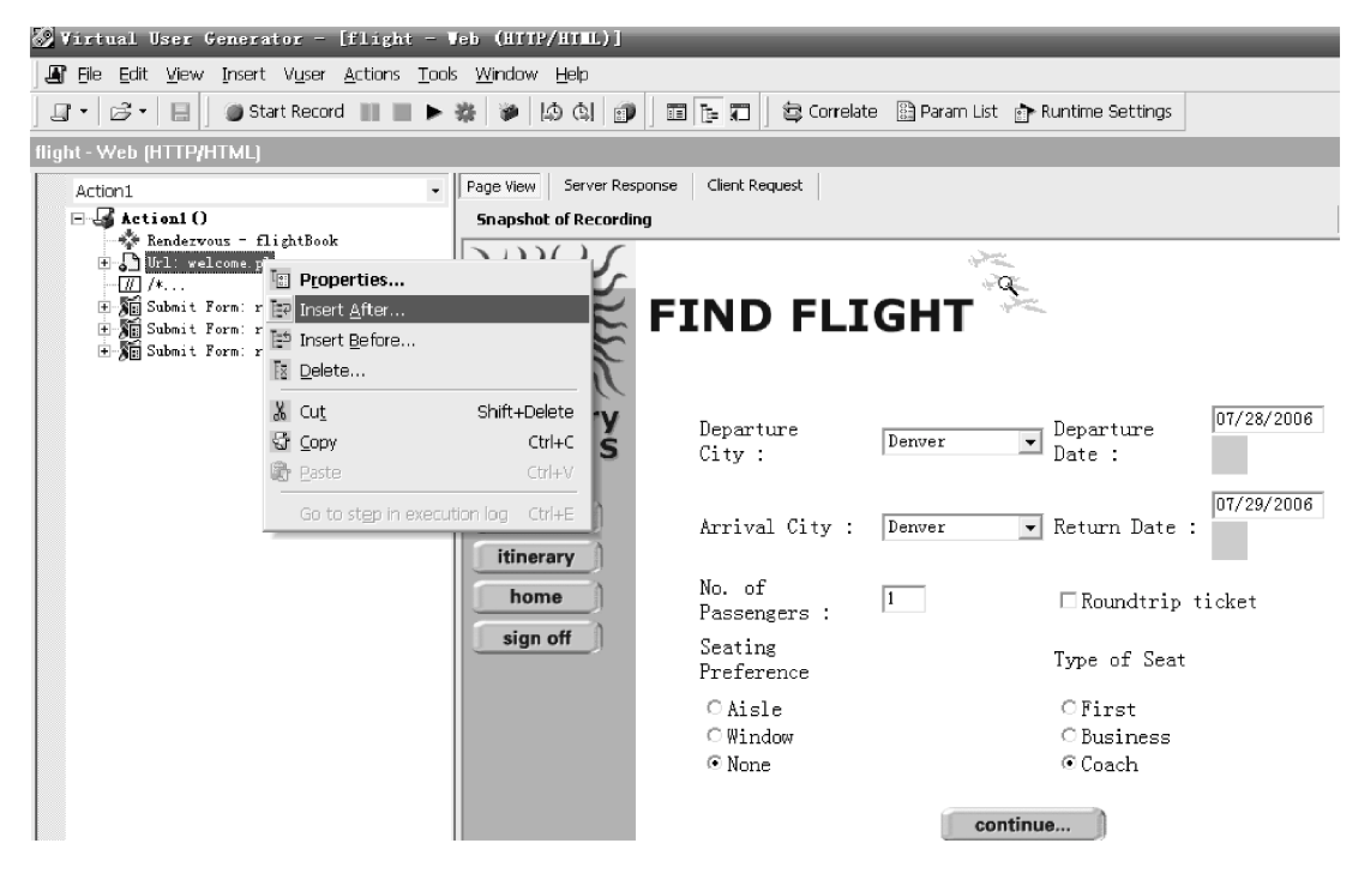


图 7.23　Virtual User Generator 主窗口中选择 Insert After

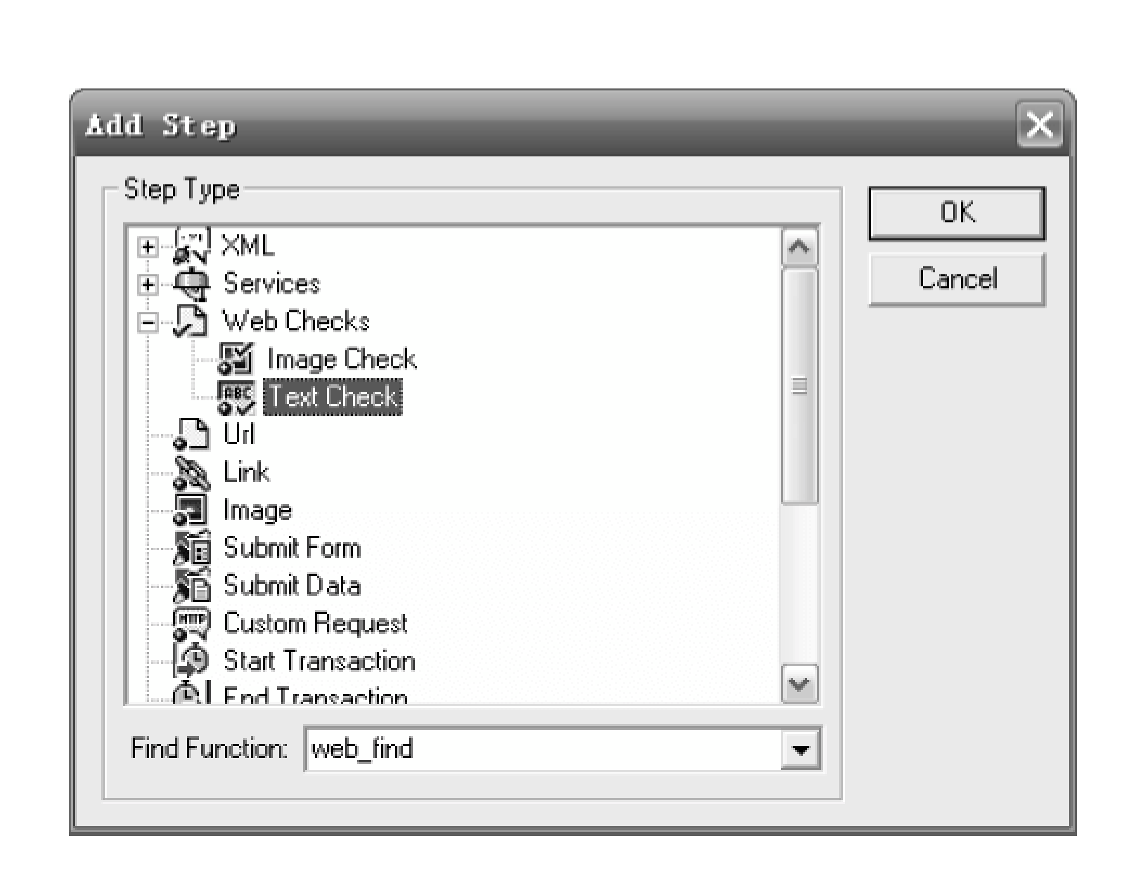


图 7.24　Add Step 窗口

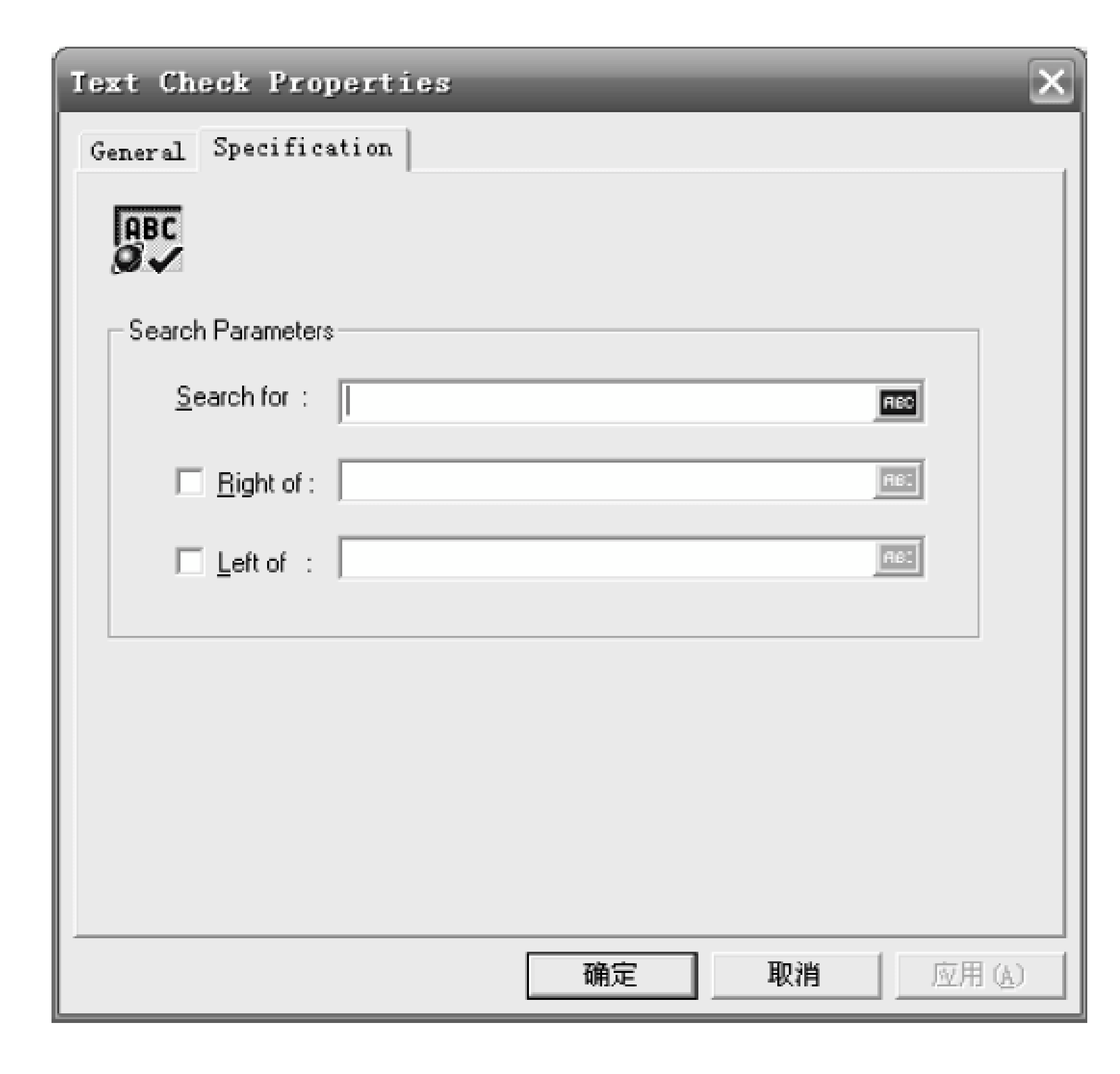


图 7.25　Text Check Properties 对话框

然后切换到【General】标签页，在【Step Name】文本框中输入操作步骤名称，添加 Text 检查点的任务即可完成，如图 7.26 所示。

添加 Imag 检查点的操作步骤和 Text 检查点差不多，这里仅仅对 Imag Check Properties 窗口进行说明：【Alternative image name】描述该图片的提示信息，【Image server file name】确定该图片的相对路径，如图 7.27 所示。

注意：如果 Web 窗体中包含有 JavaScript 脚本，那么在 TreeView 视图中显示可能会有问题。解决这个问题，可以进行如下设置。



图 7.26　Text Check Properties 窗口

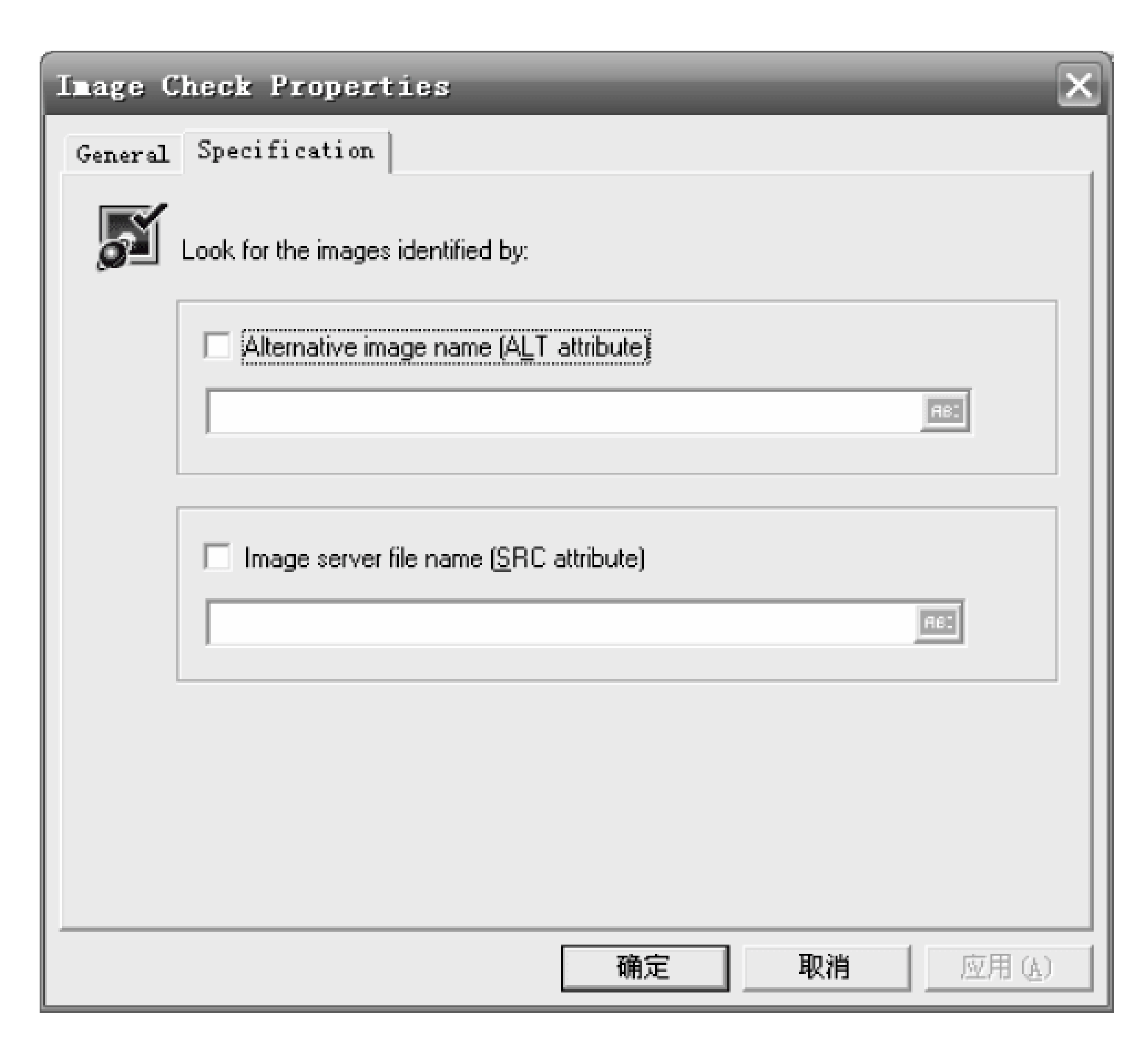


图 7.27　Image Check Properties 窗口

在菜单栏中执行【Tools】→【General Options】命令进入设置窗口，在【Correlation】标签页选中【Enable Scripting and Java applets on Snapshots viewer】即可，如图 7.28 所示。

3. 配置 Run-Time Setting

当完善了测试脚本后，需要对 VuGen 的 Run-Time Setting 进行配置。下面对经常需要设置的几个标签页进行说明。在菜单栏中执行【Vuser】→【Running Time】命令，打开 Run-Time Setting 窗口，如图 7.29 所示。

(1)【General】标签页中的【Miscellaneous】项，如图 7.29 所示。

【Error Handing】：设置 LoadRunner 在遇到错误时的处理方法，一般情况下不需要改动。

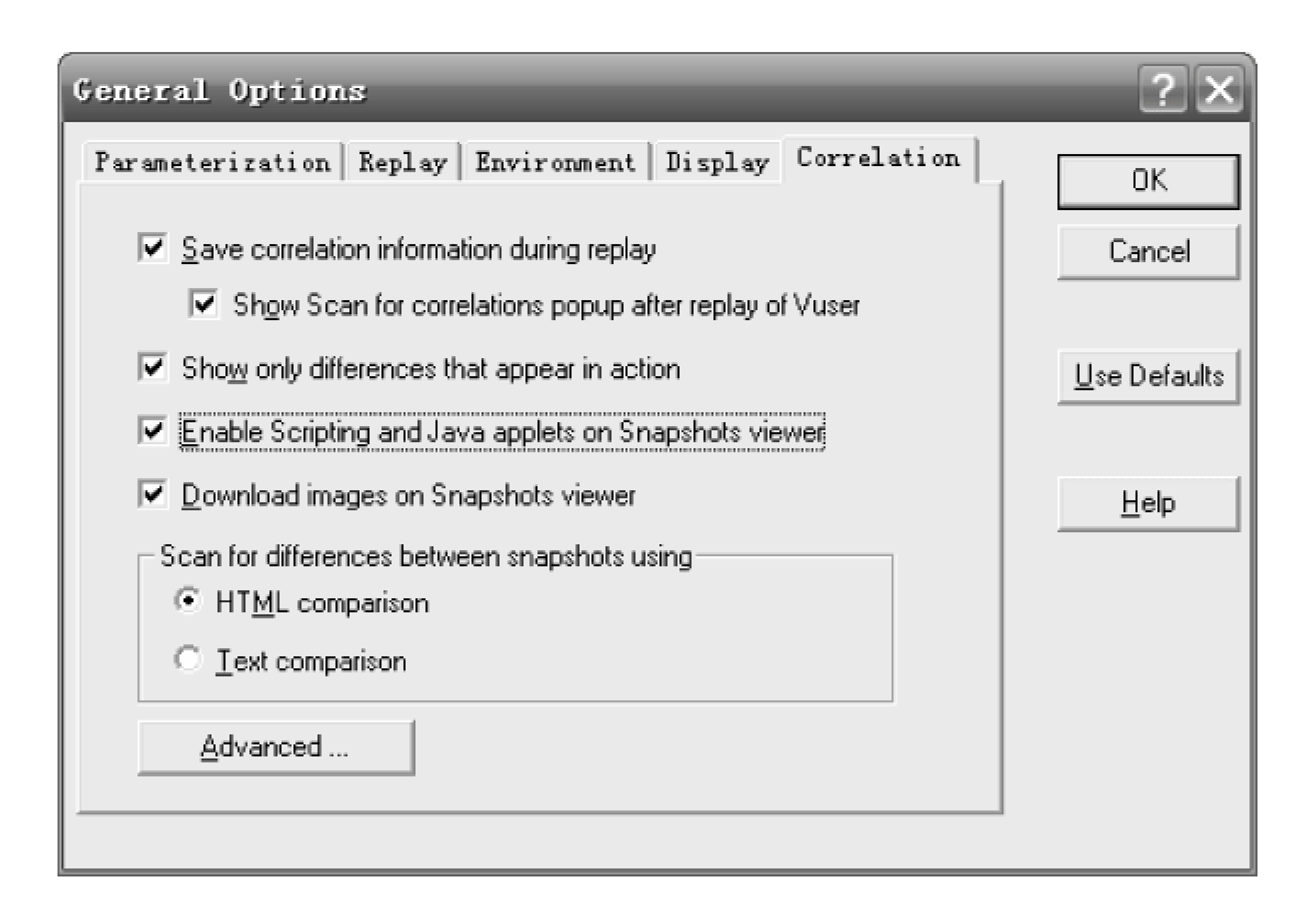


图 7.28　General Options 窗口

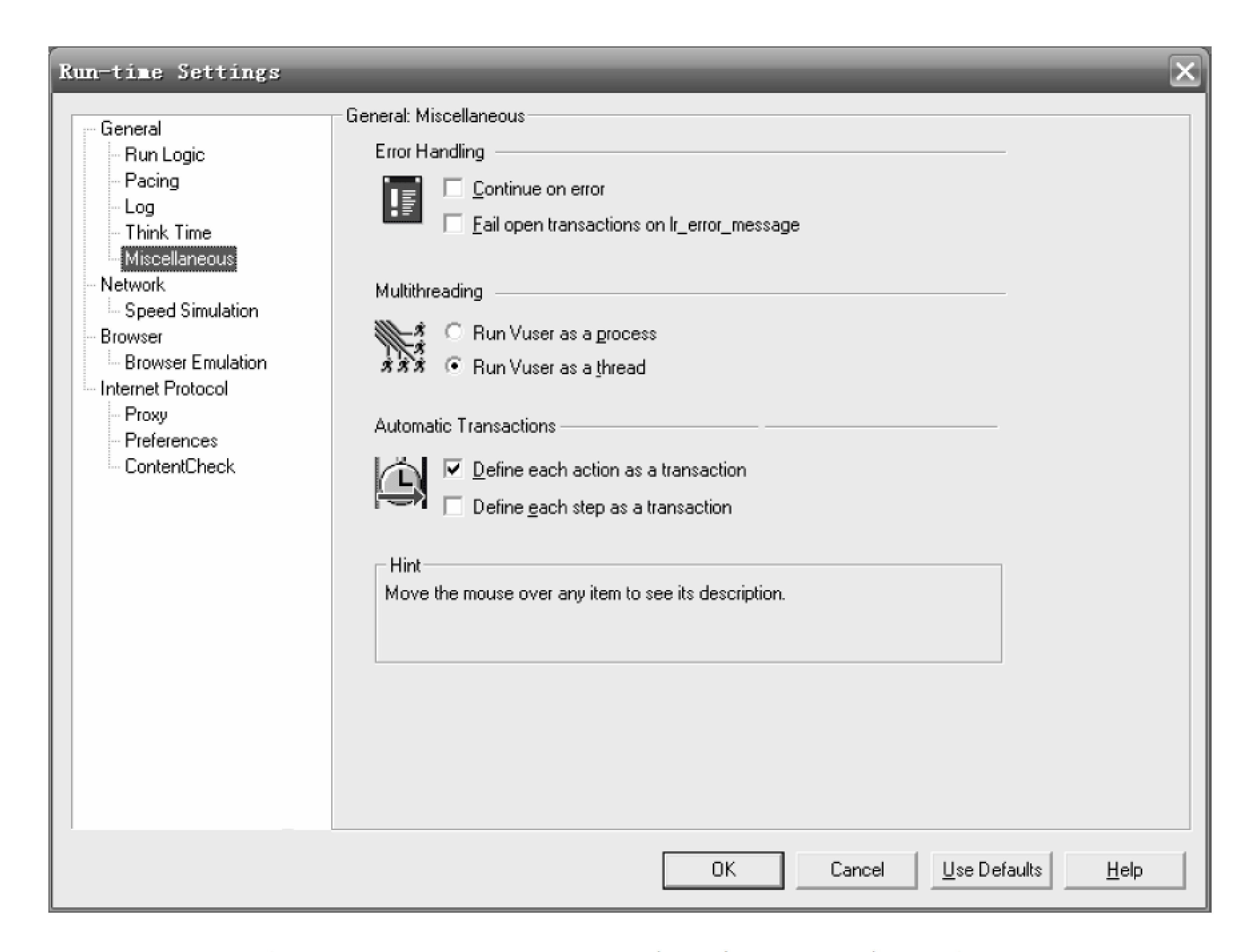


图 7.29　Run-time Settings 窗口中 General 标签页(1)

【Mutilithreading】：确定运行时为多进程还是多线程，默认是多线程，一般情况下不需要改动。

【Automatic Transactions】：默认情况下选择第一项，如果需要将脚本中的每一步都当作事务，需要选中第二项，省去添加很多次事务。

(2)【General】标签页中的【Think Time】项，如图 7.30 所示。

【Ignore think time】：选择该项，VuGen 在脚本回放过程中将不执行 lr_think_time()函数，这将给服务器造成更大的压力。

【Replay think time】：选择该项，还有四种选择。①按照录制过程中的 think time 值回放脚本；②按照录制过程中的 think time 值的整数倍回放脚本；③指定一个最小值和最大

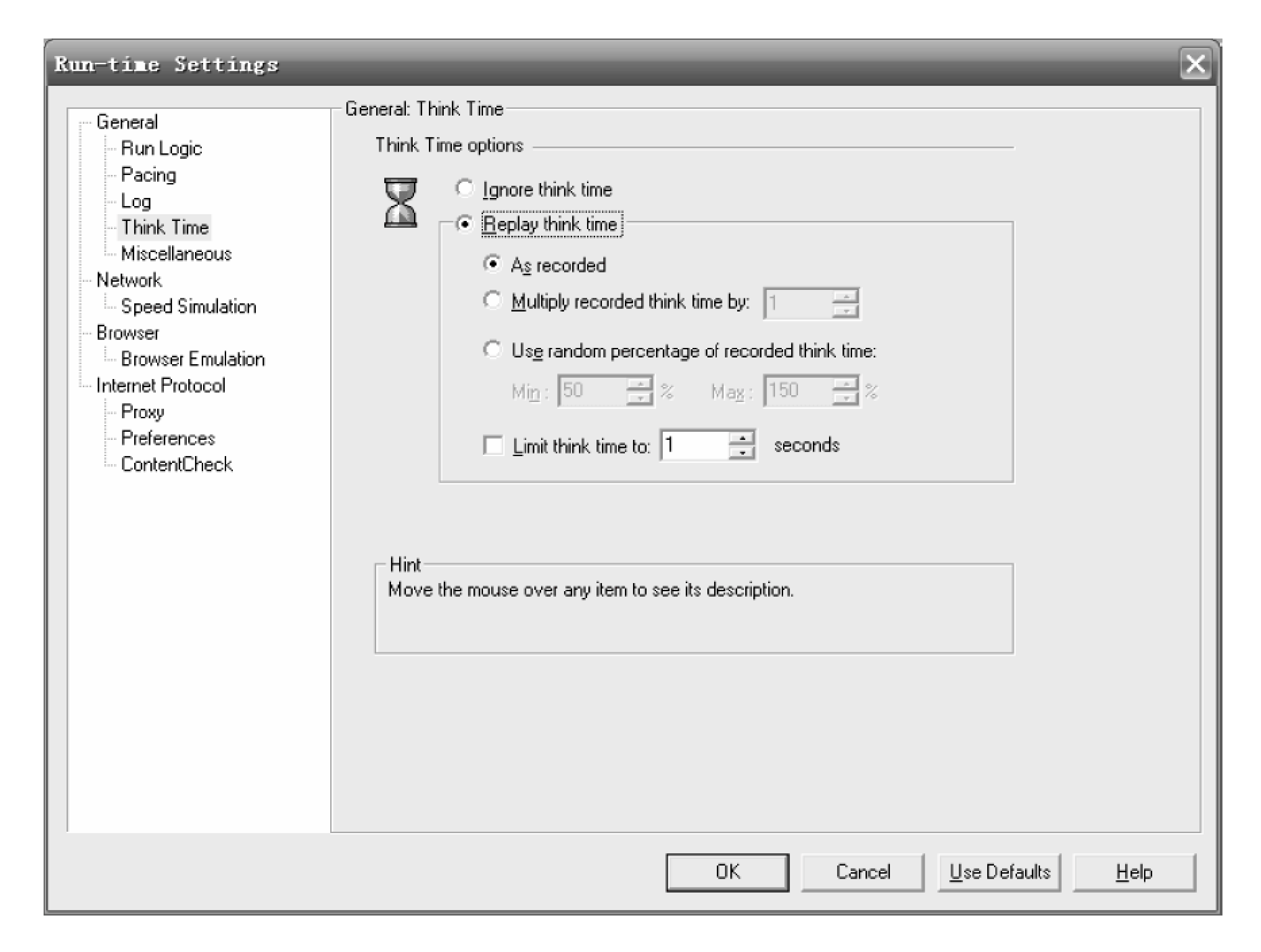


图 7.30　Run-time Settings 窗口中 General 标签页(2)

值,按照两者之间的一个随机数的值来回放脚本;④限制 think time 的最大值,这样 VeGen 在回放脚本过程中将把脚本中 think time 大于该限制值的,用该限制值来替代。

(3)【NetWork】标签页中的【Speed Simulation】项,如图 7.31 所示,选定需要的带宽方式。注意:带宽越大,给 Web 服务器造成的压力就越大。

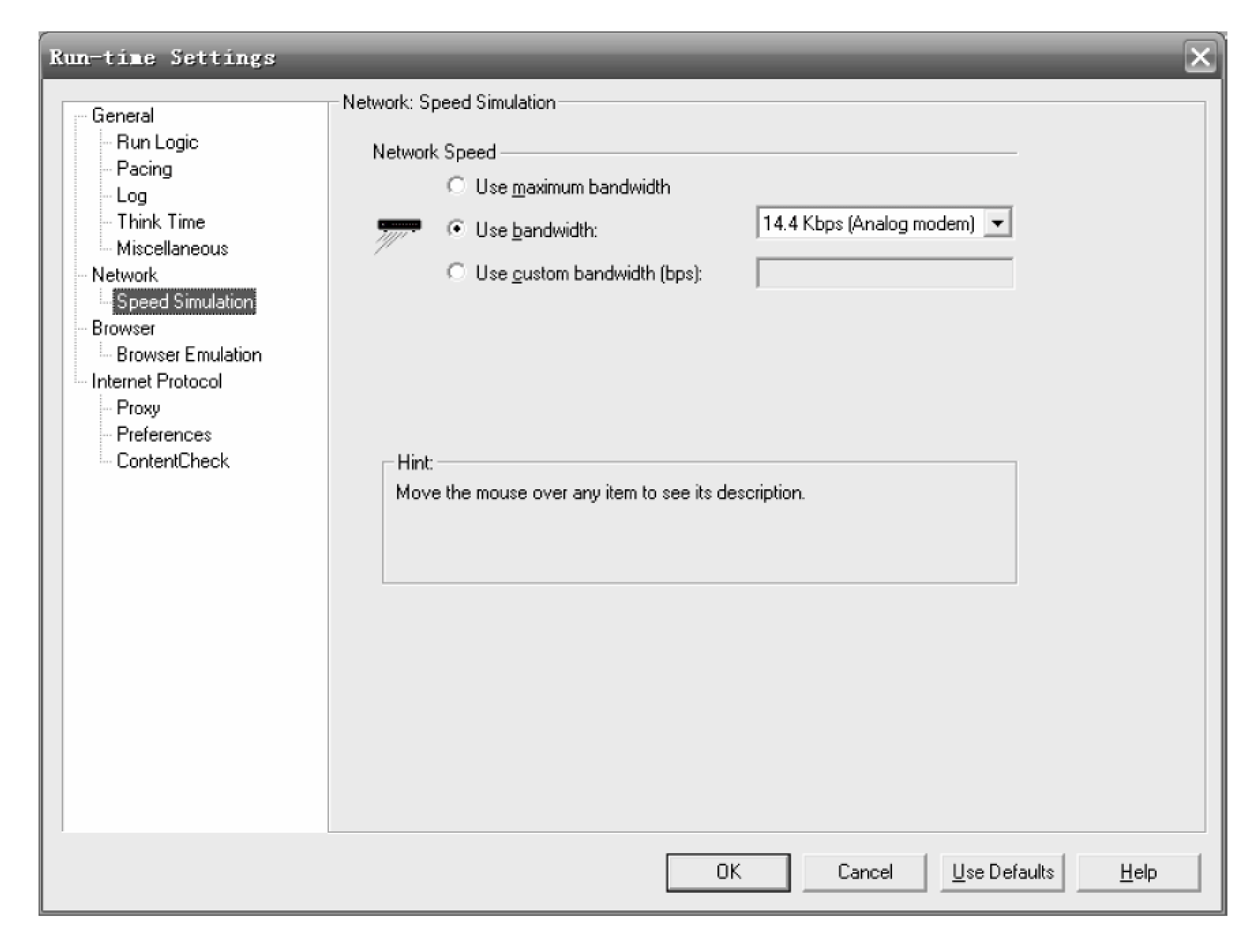


图 7.31　Run-time Settings 窗口中 NetWork 标签页

(4)【Internet Protocol】标签页中的【Preferences】项，如图 7.32 所示，如果在脚本中设置了 Image/Text 检查点，需要选中【Enable Image and text check】选项。其他选项一般情况下保持默认。

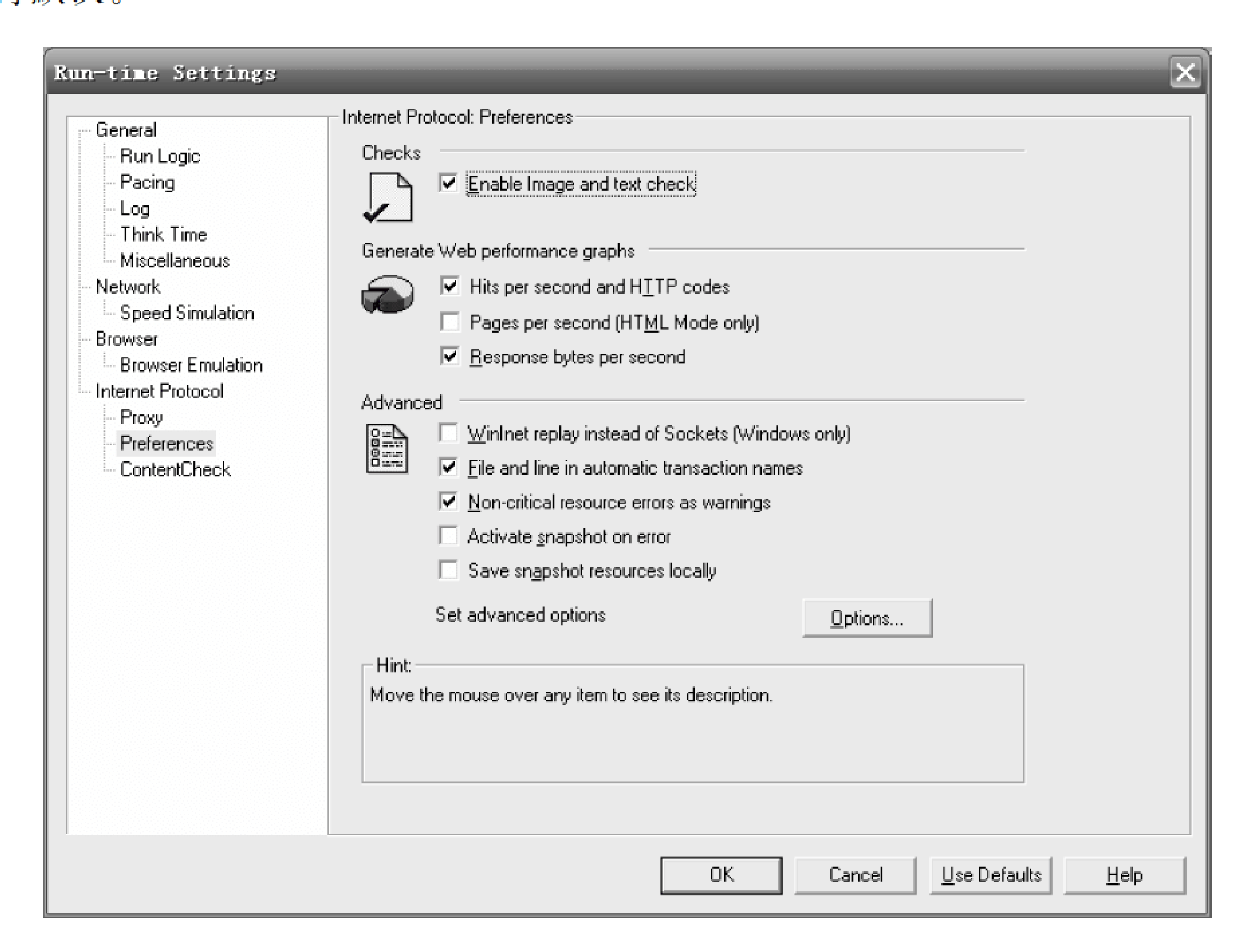


图 7.32　Run-time Settings 窗口中 Internet Protocol 标签页

(5)【Internet Protocol】标签页中的【ContentCheck】项，这里的设置是为了让 VuGen 检测何种页面为错误页面。如果被测的 Web 应用没有使用自定义的错误页面，那么这里不用作更改；如果被测的 Web 应用使用了自定义的错误页面，那么这里需要定义，以便让 VuGen 在运行过程中检测，服务器返回的页面是否包含预定义的字符串，进而判断该页面是否为错误页面。如果是，VuGen 就停止运行，指示运行失败。

4. 运行测试脚本

经过以上的各个步骤后，脚本就可以运行了。在菜单栏中选择【Vuser】→【Run】命令，或者在工具栏中直接单击【运行】按钮。

执行【运行】命令后，VuGen 先编译脚本，检查是否有语法等错误，如果有错误，VuGen 将会提示错误，双击错误提示，VuGen 能够定位到出现错误的那一行。为了验证脚本的正确性，还可以调试脚本，比如在脚本中加断点等，操作和在 VC 中完全一样，如图 7.33 所示。

如果编译通过，就会开始运行，然后会出现运行结果，如图 7.34 所示。

5. VuGen 其他有用的功能

(1) 压缩脚本文件

在菜单栏执行【File】→【Zip and Email】命令：把脚本所有文件压缩并作为附件发送邮件；

在菜单栏执行【File】→【Export to ZipFile】命令：压缩脚本所有文件到一个 Zip 文件。

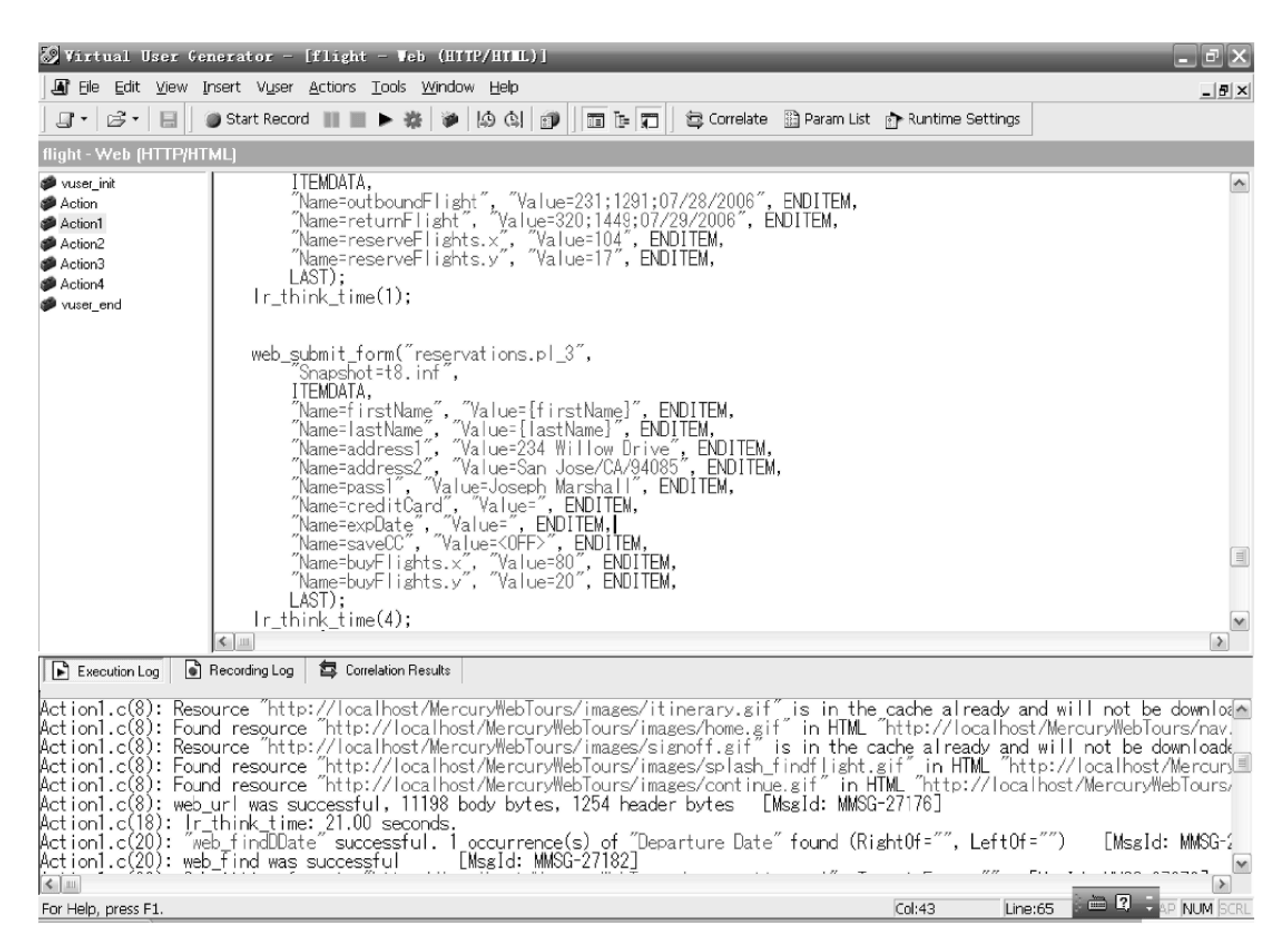


图 7.33　Virtual User Generator 脚本窗口

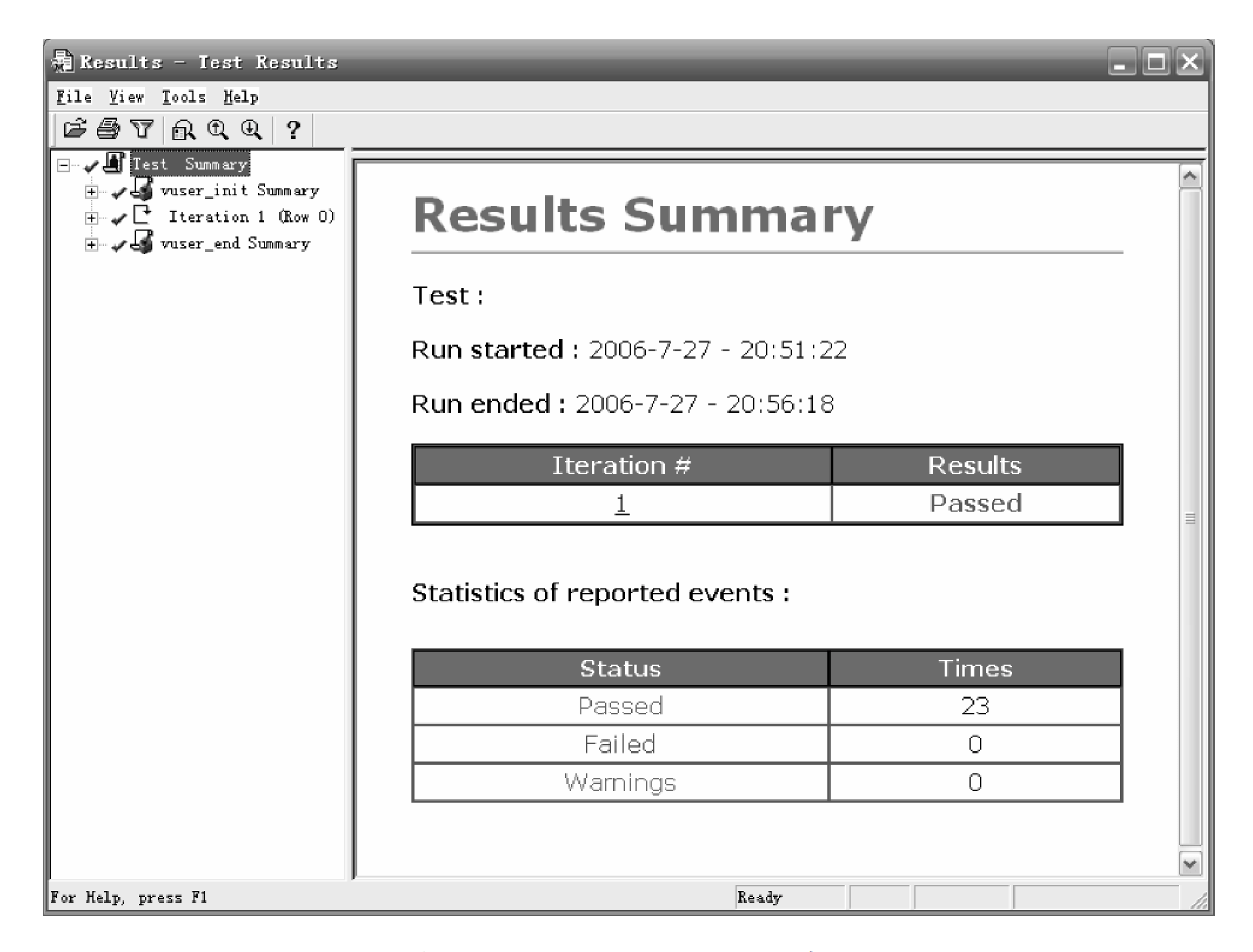


图 7.34　Results-Text Results 窗口

（2）tools 菜单

【TestDirector Connection】：连接 TD 数据库可以把脚本添加到 TD 数据库中。

【Create Controller Scenario】：启动 Controller，并把当前脚本添加到场景中。

【Compare With Vuser】：比较两个脚本。

7.3.3 创建运行场景

运行场景描述在测试活动中发生的各种事件。一个运行场景包括一个运行虚拟用户活动的 Load Generator 机器列表，一个测试脚本的列表以及大量的虚拟用户和虚拟用户组，使用 Controller 来创建运行场景。

在开始菜单中，启动 Controller 程序，出现 New Scenario 窗口，如图 7.35 所示。如果没有出现，可以在菜单中执行【File】→【New】命令，或者在工具栏中单击【New】按钮。

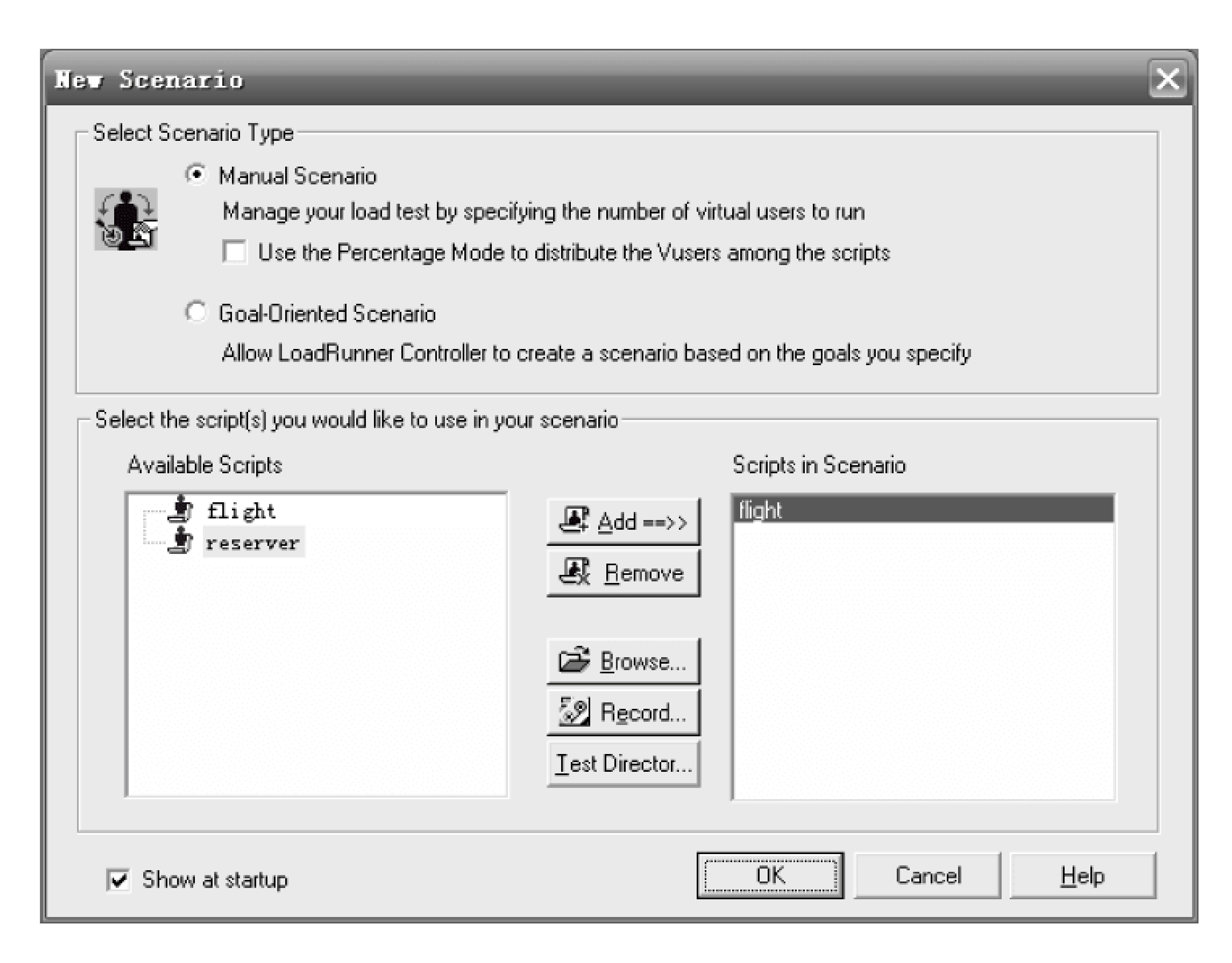


图 7.35　New Scenario 窗口

在新建场景的窗口，选择一种场景类型。下面对 3 种类型进行简单的说明。

（1）Manual Scenario：该项要完全手动的设置场景。

（2）Manual Scenario with Percentage Mode：该项只有在"Manual Scenario"选中的情况下才能选择。选择该项后，在场景中需要定义要使用的虚拟用户的总数，Load Generator machine 机器集，然后为每一个脚本分配要运行的虚拟用户的百分比。

（3）Goal-Oriented Scenario：在测试计划中，如果测试计划是要达到某个性能指标，比如，每秒多少点击，每秒多少 transactions，能到达多少 VU，某个 Transaction 在某个范围 VU(500～1000)内的反应时间等，选择该项，LoadRunner 将基于这个目标，自动创建一个场景。在场景中，只要定义好目标即可。

1. 选择场景类型为 Manual Scenario

（1）选择 Vuser Groups

在可选脚本框中，选中需要的脚本，单击【Add】按钮将其添加到场景操作中。或者单击

【Browse】按钮选择脚本的路径，选择想要添加的脚本。

如果需要在已经打开的场景中添加脚本，直接单击【ScriptPath】栏下的下拉菜单，或者单击右边的【Add Group】按钮。

(2) 添加 Load Generator Machines

单击右边的【Generators】按钮，出现 Load Generators 窗口，如图 7.36 所示。

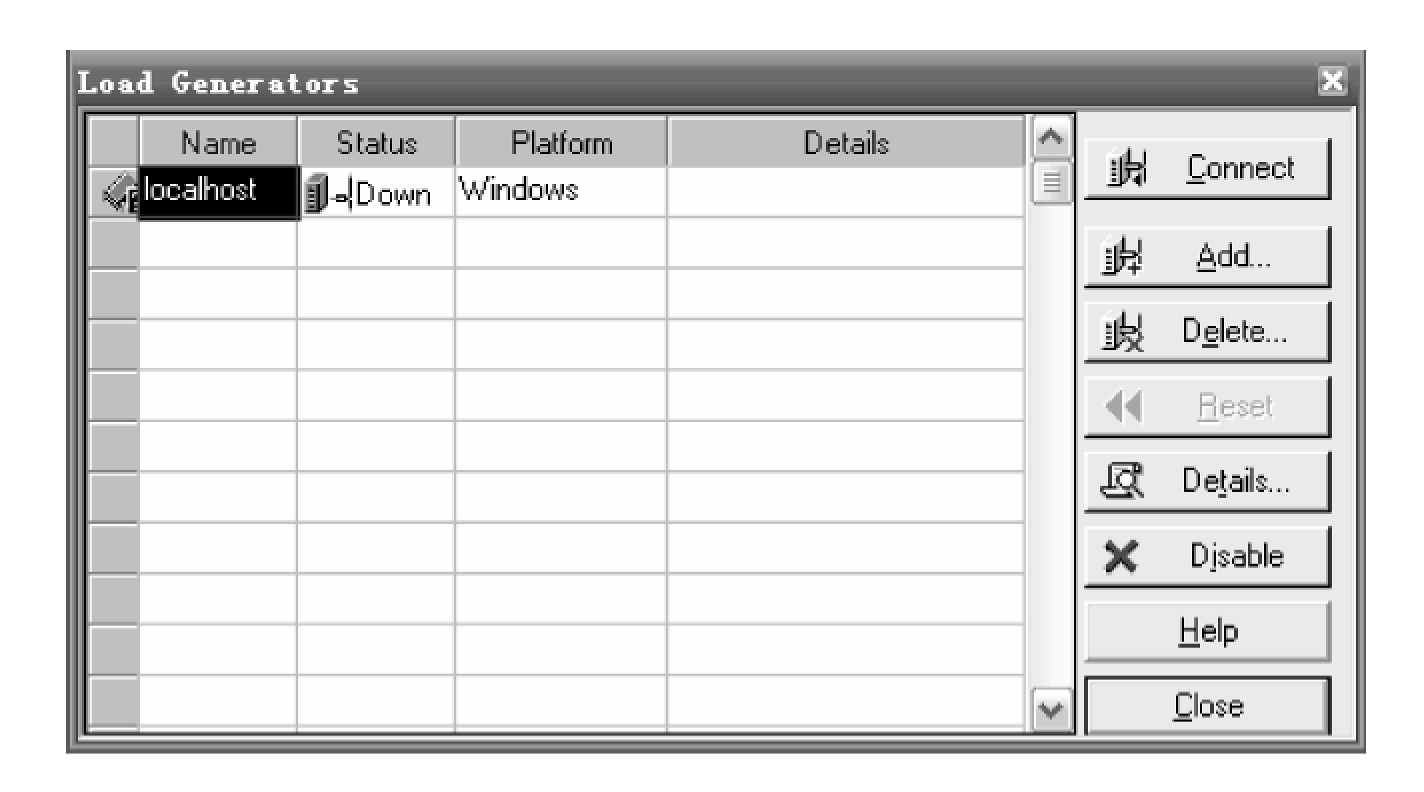


图 7.36　Load Generators 窗口(1)

添加 LoadGenerator 后，执行【Connect】操作，若 Status 为 Ready，如图 7.37 所示，表示该机器连接正常，如果为 Failed，表示该机器不能连接，请检查原因。可以把这个列表保存下来，执行菜单命令【Scenario】→【Save Load Generator List As Default】即可。

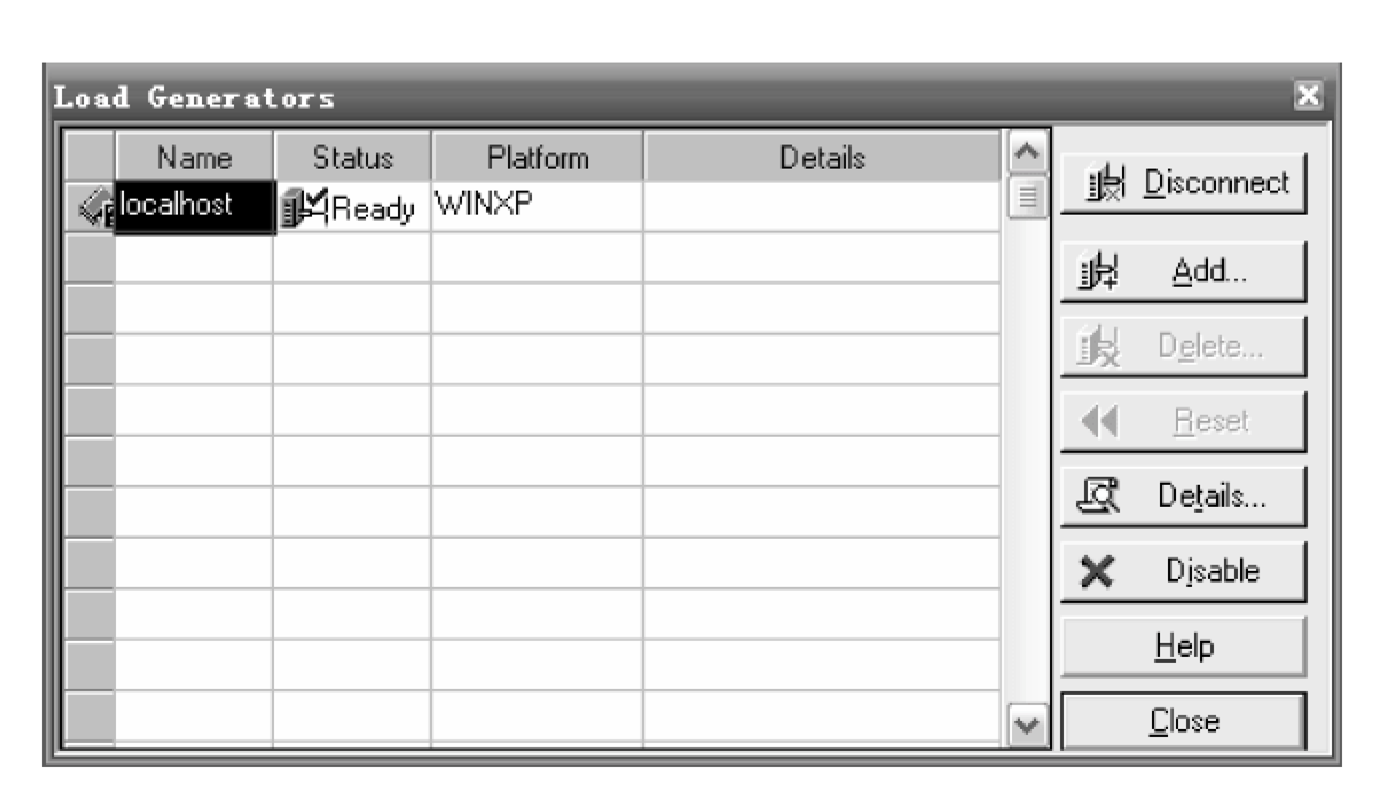


图 7.37　Load Generators 窗口(2)

(3) 添加虚拟用户

首先单击【Quantity】栏下的文本框设置虚拟用户数，如图 7.38 所示。

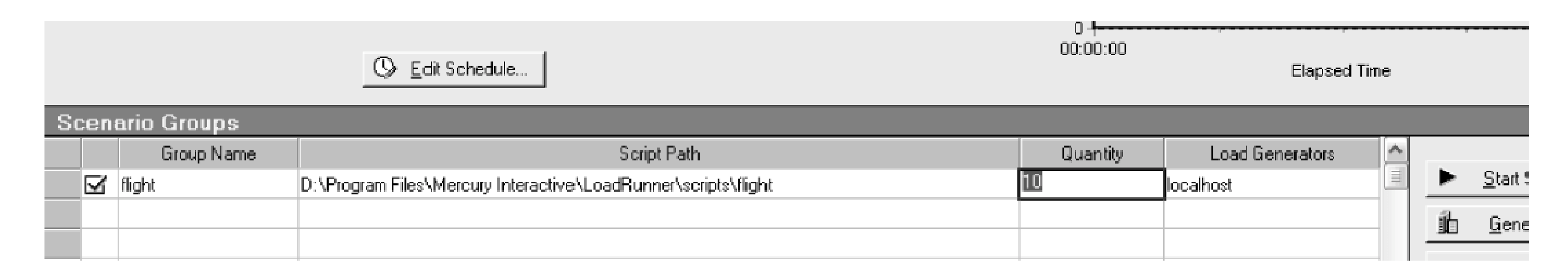


图 7.38　设置虚拟用户数

再单击右边的【VUsers】按钮设置该虚拟用户将在哪个 Load Generators 上运行。

(4) 设置 Schedule

这里的设置是 3 种场景类型最重要的区别之处，单击【Edit Schedule】按钮，即可进入 Schedule Builder 设置窗口。

① 【Ramp up】标签页，如图 7.39 所示。

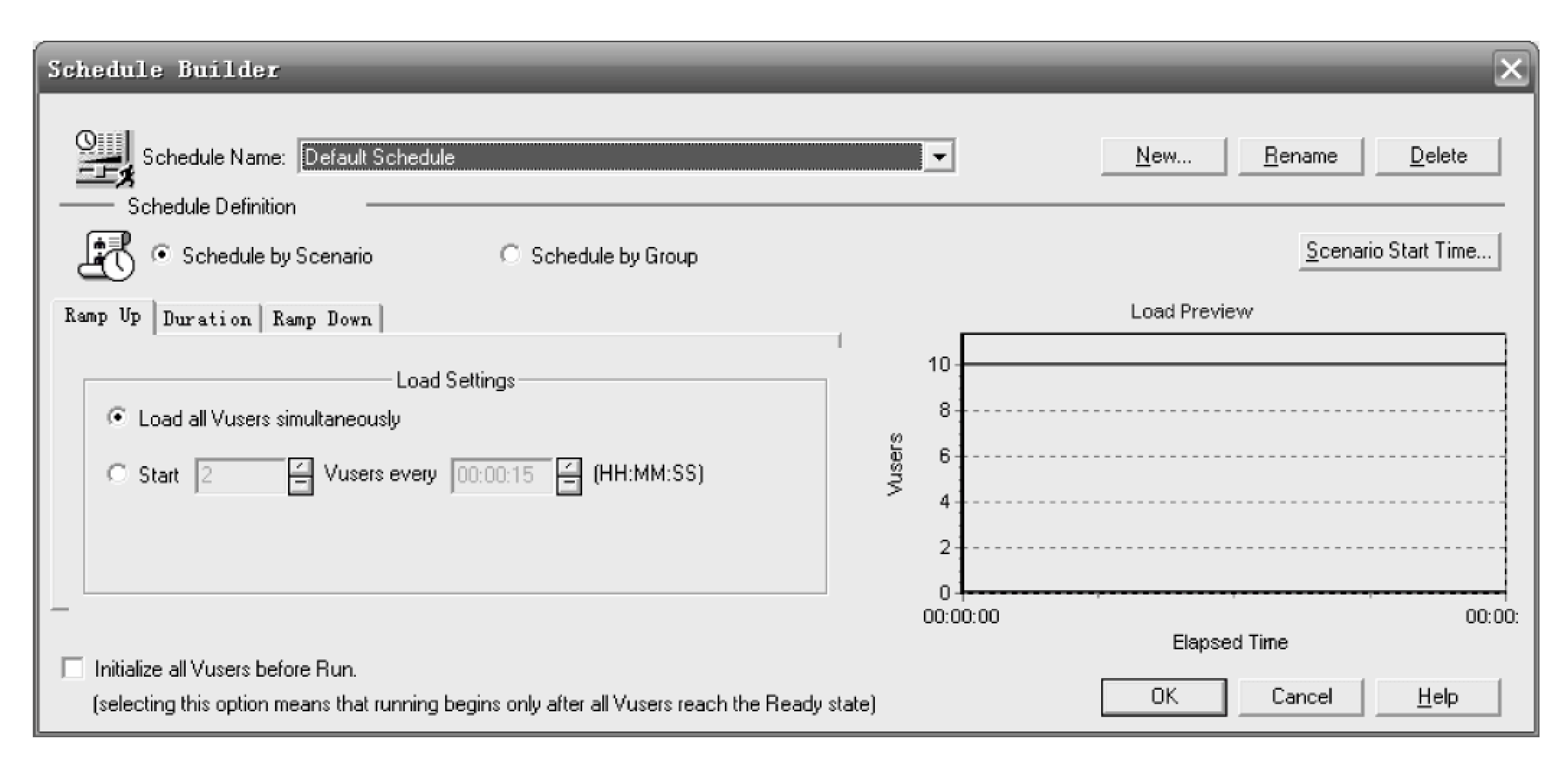


图 7.39 Schedule Builder 窗口(1)

【Load all Vusers simultaneously】：表示同时加载所有的用户。

【Start... Vuvsers Every...】：表示每多少时间加载多少用户，时间和用户数由自己来定义。

② 【Duration】标签页，如图 7.40 所示。

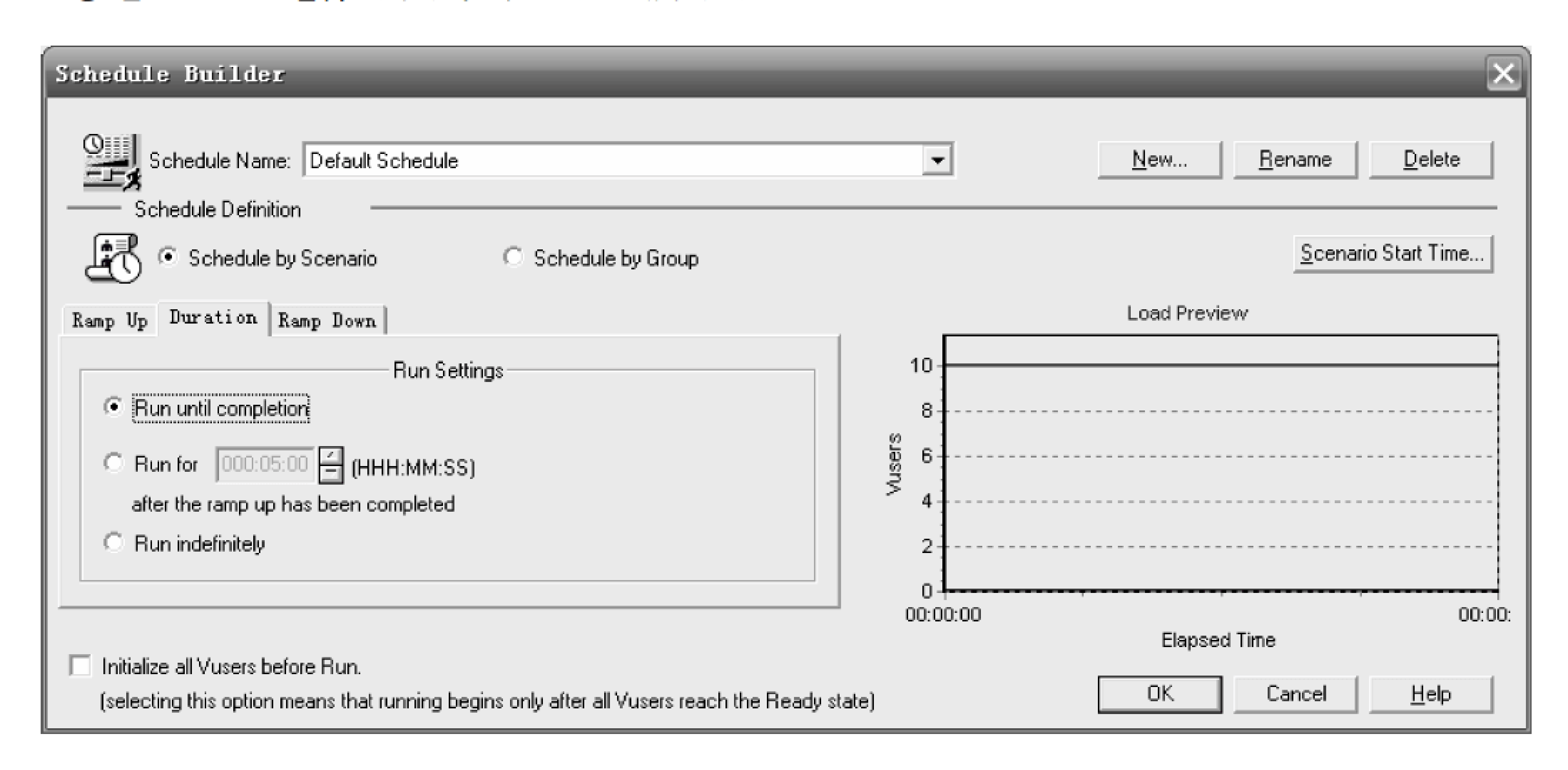


图 7.40 Schedule Builder 窗口(2)

【Run until completion】：虚拟用户运行一遍，场景就停止。

【Run for...after the ramp up has been completed】：加载完所有用户后，场景继续运行

一段时间后停止，这个时间由自己定义。

【Run indefintely】：场景一直运行，不停止。

③ 在选中【Run for…after the ramp up has been completed】项时，才对【Ramp Down】标签页设置，如图 7.41 所示。

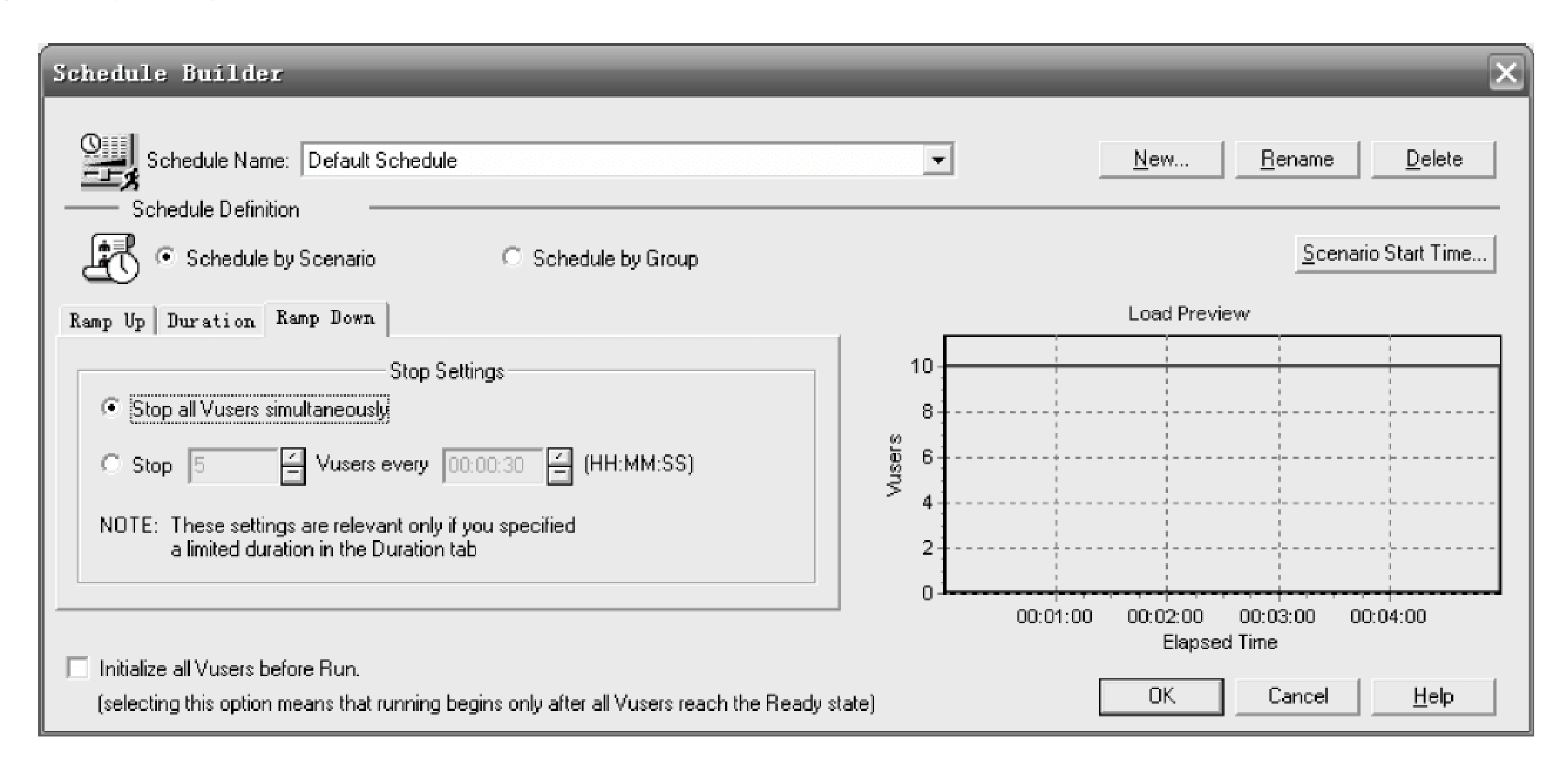


图 7.41　Schedule Builder 窗口(1)

【Stop all Vusers simultaneously】：同时停止所有的用户。

【Stop… Vuvsers Every…】：每隔多少时间停止多少用户，时间和用户数由自己来定义。

④ 单击【Scenario Start Time】按钮，进入 Scenario Start 窗口，如图 7.42 所示。

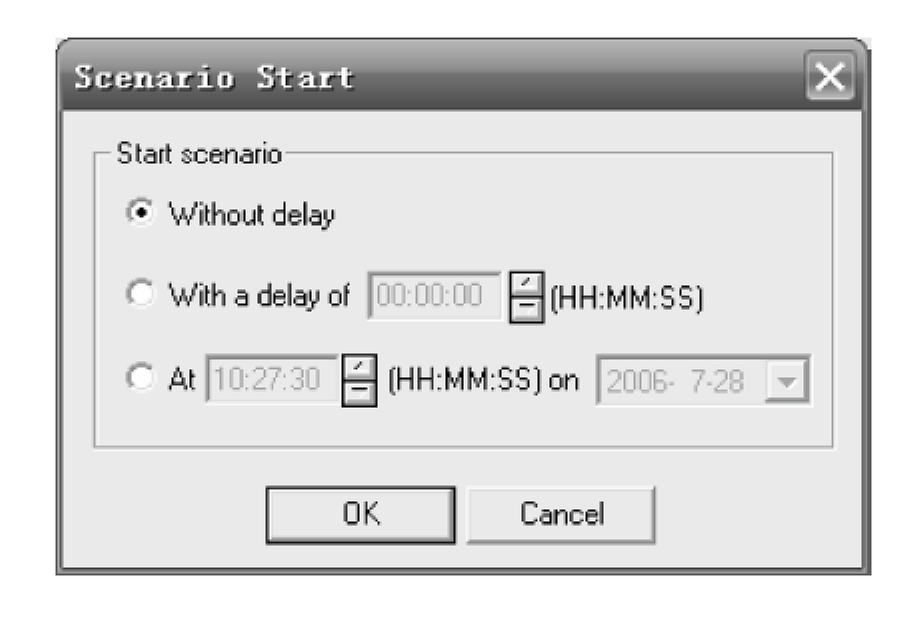


图 7.42　Scenario Start 窗口

【Without delay】：表示 Start Scenario 后，场景开始运行。

【With a delay of…】：表示 Start Scenario 后，场景延迟指定的时间后开始运行。

【At… On…】：表示 Start Scenario 后，场景在指定时刻开始运行。

(5) 设置集合点

如果在脚本中设置了集合点，还需要在 Controller 中设置集合点策略。

在菜单中执行【Scenario】→【Rendezvous】调出设置集合点策略的窗口。如果在脚本中没有设置集合点，菜单中【Scenario】下的【Rendezvous】是灰色的，不可用。

如图 7.43 所示，单击【Policy】按钮，进入策略设置窗口选中需要的选项，单击【OK】按钮保存设置并退出，如图 7.44 所示。

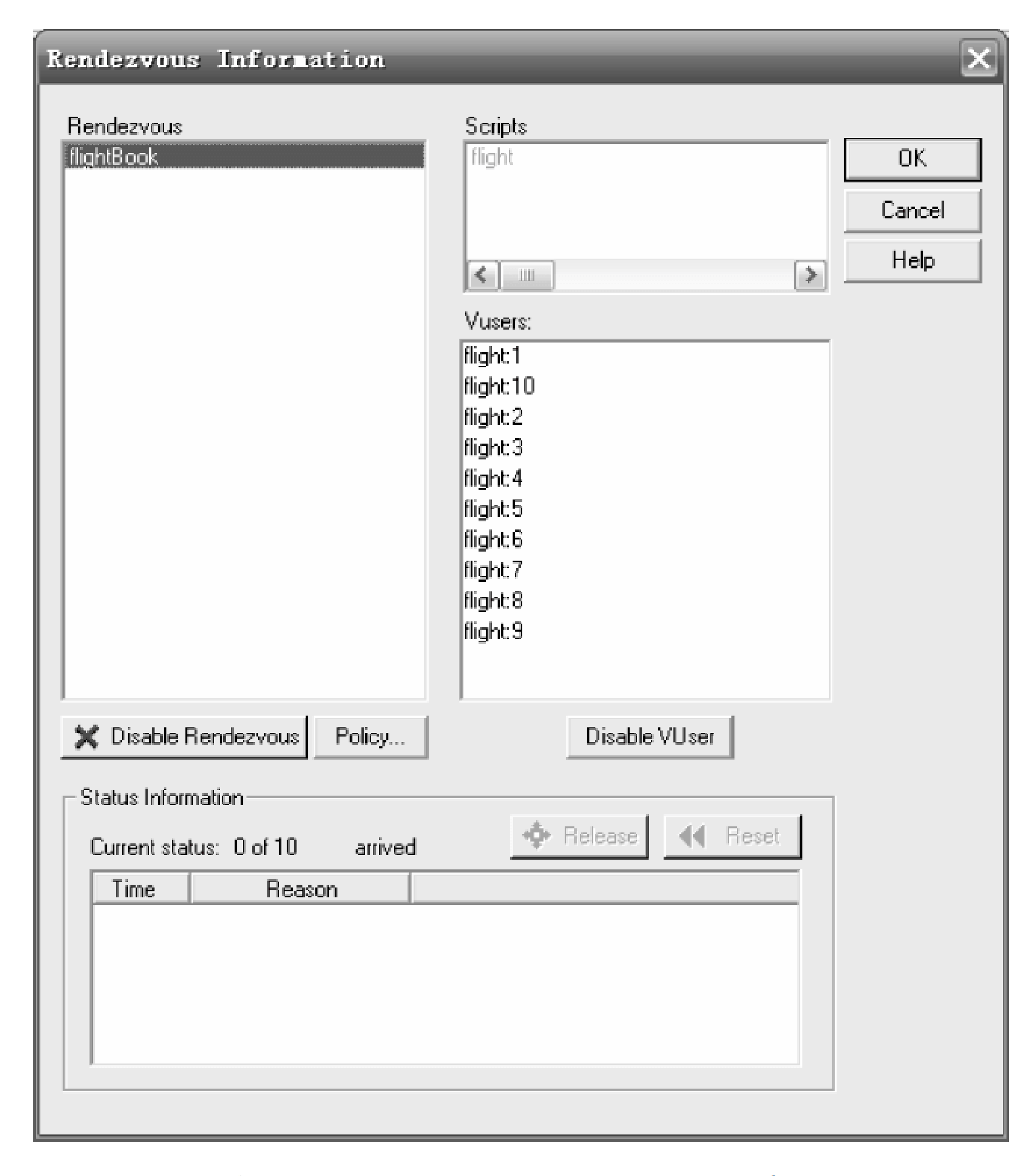


图 7.43　Rendezvous Information 窗口

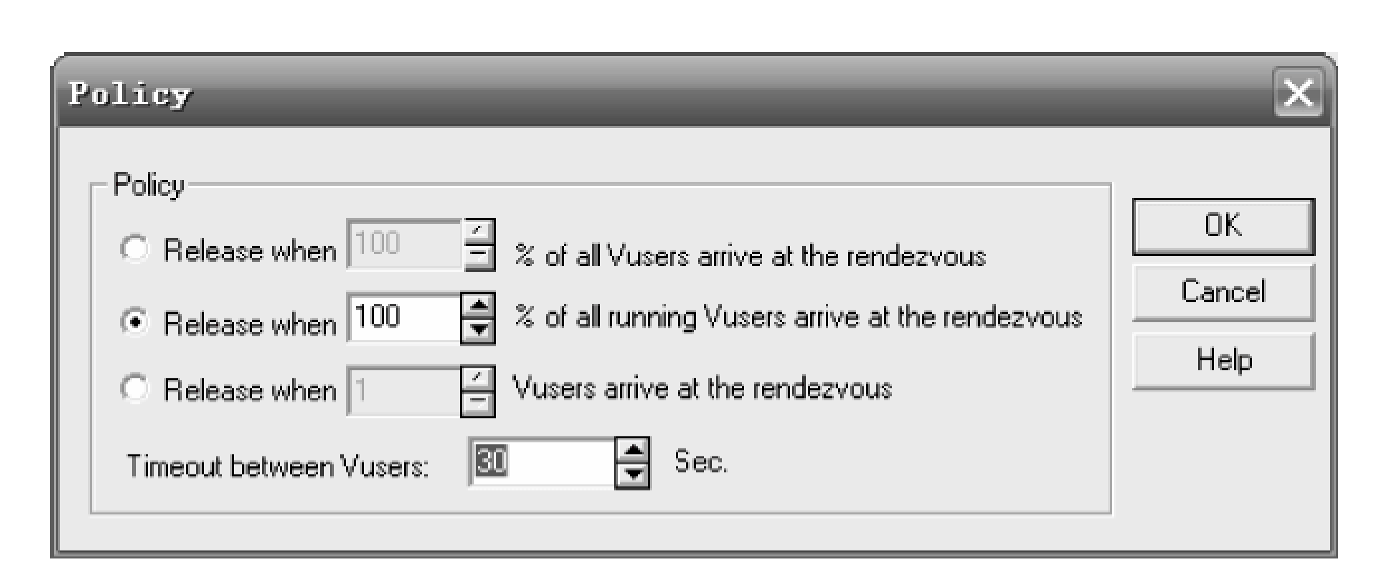


图 7.44　Policy 窗口

(6) 设置结果文件保存路径

在菜单中执行【Results】→【Results Settings】命令，调出结果文件的保存路径，该路径最好在每次场景运行前重新设置一下，如图 7.45 所示。

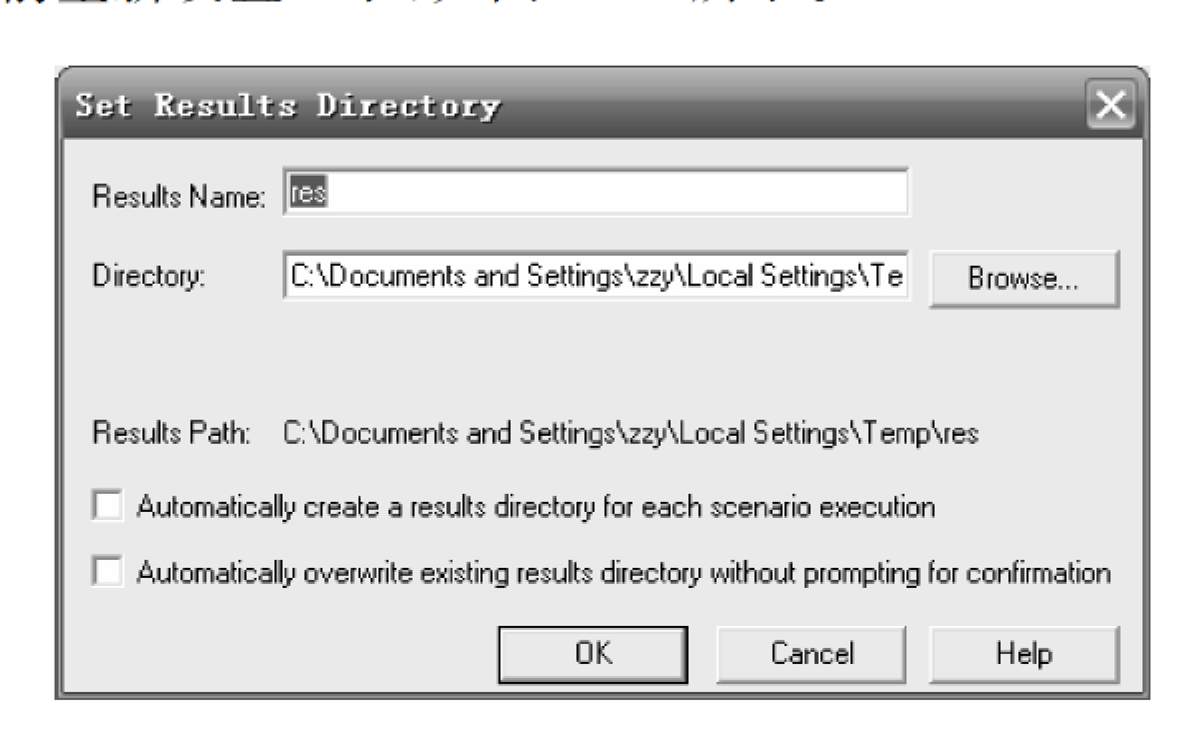


图 7.45　Set Results Directory 窗口

(7) Run-Time Setting

请参考 7.3.2 节。

2. 选择场景类型为 Manual Scenario with Percentage Mode

该场景类型和“Manual Scenario”类型类似，如图 7.46 所示，下面对它们不一样的地方进行设置。

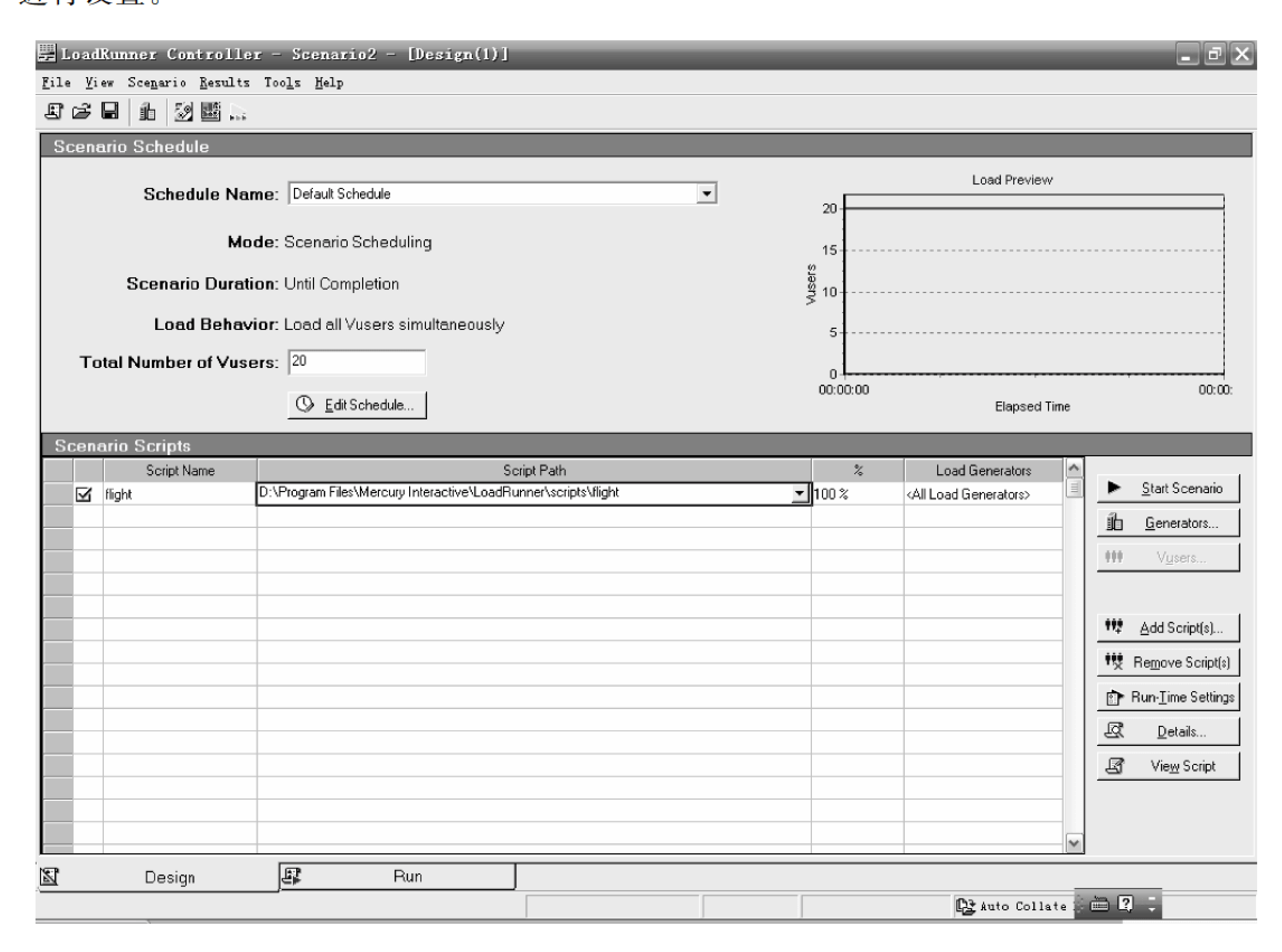


图 7.46　Manual Scenario with Percentage Mode 场景

【Total Number of Vusers】：填写虚拟用户总数。

【Scenario Script】标签页的【%】：填写用户数的百分比。

3. 选择场景类型为 Goal-Oriented Scenario

同样，只对不同的地方进行设置讲解。单击【Edit Scenario Goal】按钮，编辑该场景的目标。

(1) 在【Goal Profile Name】中选择目标的种类每次场景运行只能设置一个目标。下面是简单目标的种类。

① Virtual Users Goal。如果需要测试多少人可以同时运行 Web 应用，那么推荐定义 Virtual Users Goal。运行定义该目标类型的场景和运行 Manual 类型的场景类似。

② Hits per Second。如果想测试 Web Server 的真正实力，推荐定义目标类型为：Hits per Second、Pages per Minute 或者 Transactions per Second，这些类型都需要指定一个虚拟用户的最小值和最大值的范围。

Controller 试图使用最少的虚拟用户来达到定义的目标。如果使用最少的用户不能达

到目标，Controller 增加用户数，直到定义的最大值。如果使用了最多的虚拟用户数，定义的目标还没有实现，那么需要增加最大用户数，重新执行场景。

③ Transactions per Second。类型需要脚本中包含有事务。

④ Transactions Response Time。如果想知道在多少用户并发访问网站时，事务的响应时间达到性能指标说明书中规定响应时间的最大值，那么推荐使用 Transactions Response Time 类型。指定需要测试的事务的名称，虚拟用户数量的最小值和最大值，还有预先定义好的事务的响应时间。

在场景运行中，如果使用了最多的虚拟用户，还不能达到定义的最大响应时间，说明 Web Server 还有能力接纳定义的虚拟用户的最多数量；如果使用了部分虚拟用户，就达到了定义的最大的响应时间，或者 LoadRunner 提示如果使用最多数量的虚拟用户时将要超过最大响应时间，那么需要重新设计或者修补应用程序，同时可能需要升级 Web Server 的软硬件。

(2)【Scenario Settings】标签页，如图 7.47 所示。

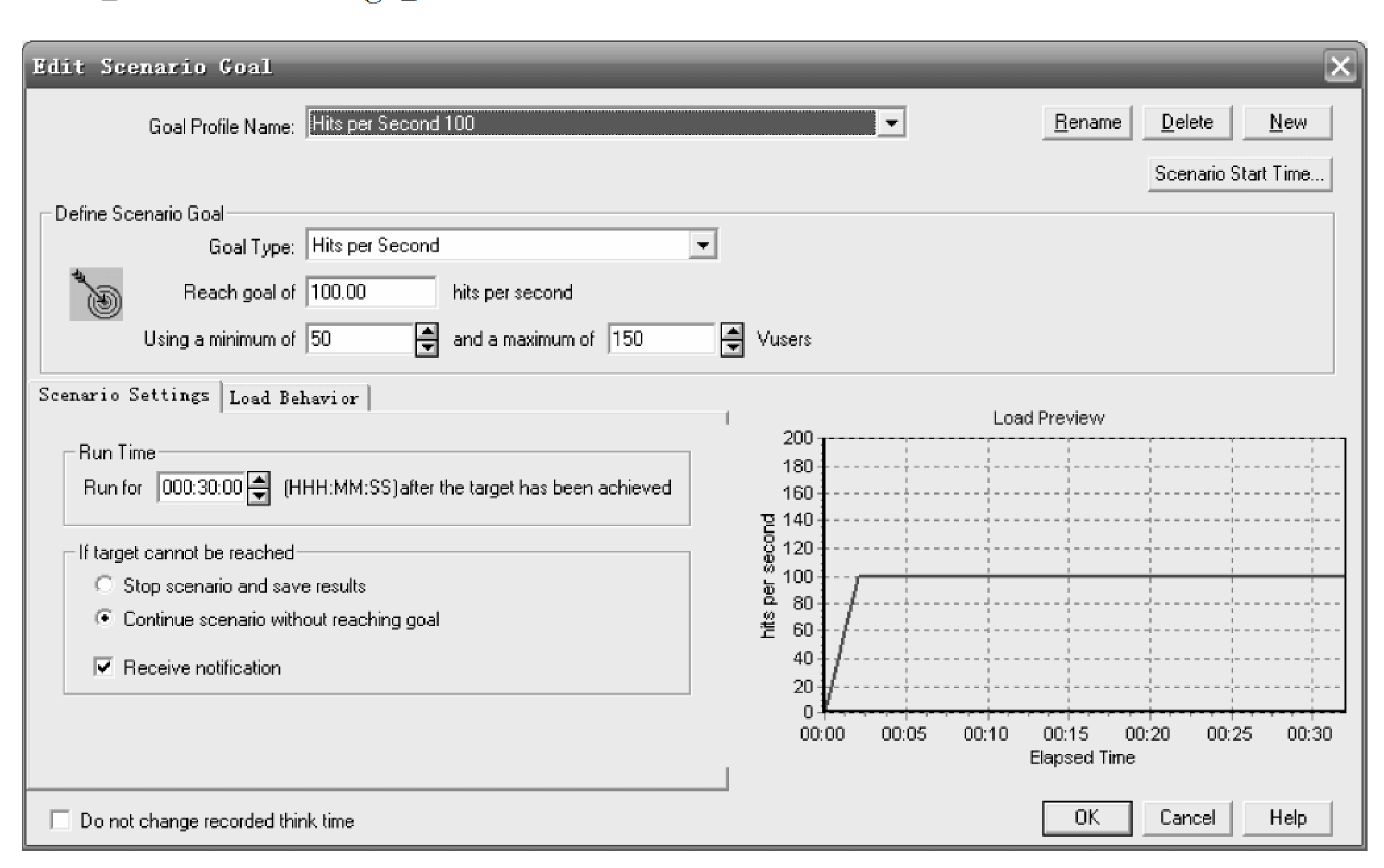


图 7.47　Edit Scenario Goal 标签页 Scenario Settings

【Run Time】：定义达到目标后场景继续运行的时间。

【If target cannot be reached】：定义如果目标不能达到是如何操作，①停止执行，保存结果；②继续执行，直到达到目标为止。

(3)【Load Behavior】标签页，如图 7.48 所示。

【Automatic】：自动让 Controller 自动加载虚拟用户。

【Reach target number of...hits per second after...】：定义每个时间内需要达到目标的虚拟用户数。

【Step up by... hits per second every...】：定义按照时间分步加载虚拟用户，时间和用户数自己定义。

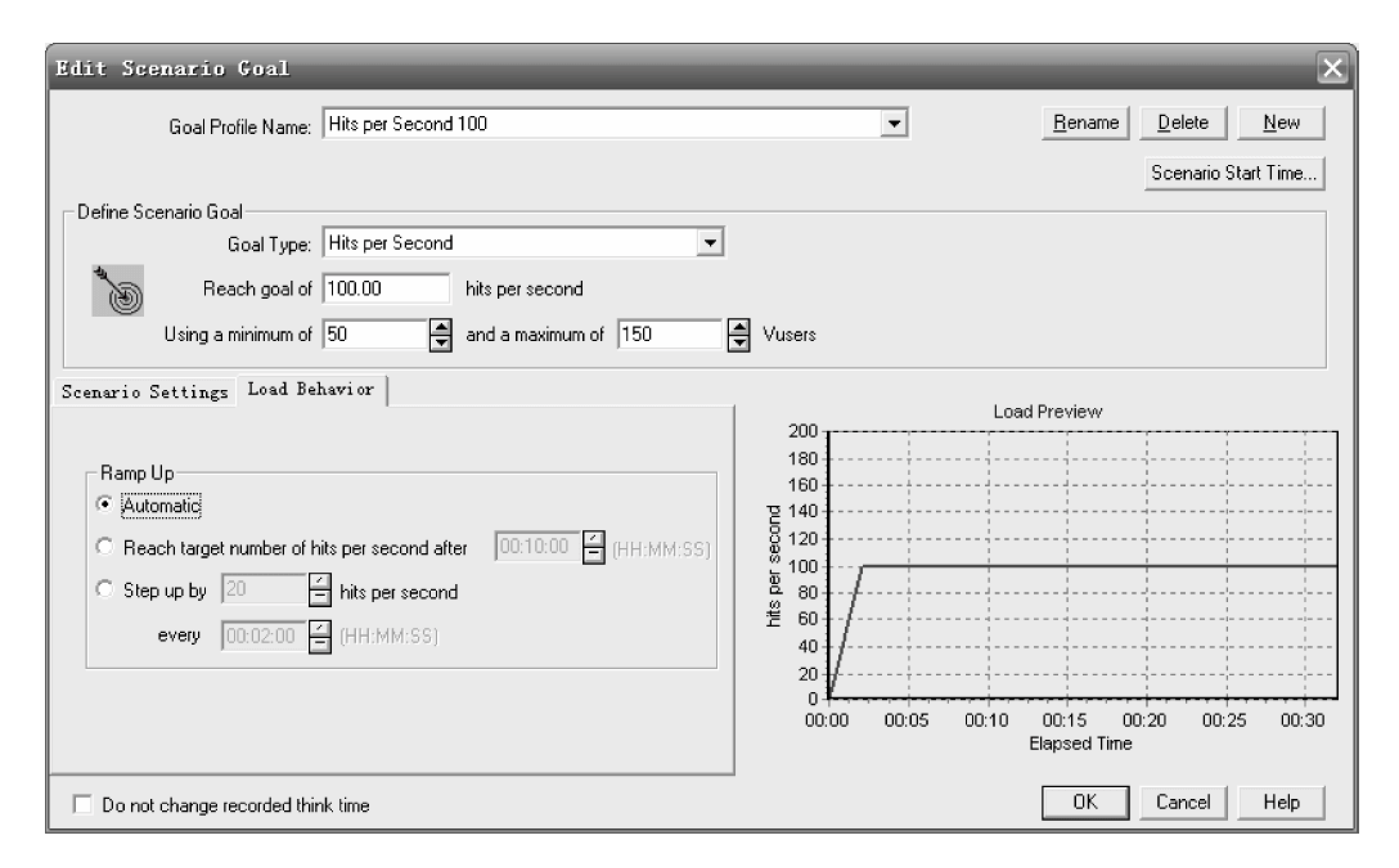


图 7.48　Edit Scenario Goal 标签页 Load Behavior

4. 其他有用的设置

(1) 场景类型的转化

在菜单栏中选择【Scenario】→【Convert Scenario into the Percentage Mode】命令，使用这个选项，可以在 Percentage Mode 和 Vusers Group 之间互相转化，不过一些设置可能会丢失。更详细的信息请参考帮助文档。

(2) 启用 IP Spoofer(IP 欺骗)

当运行场景时，虚拟用户使用它们所在的 Load Generator 的固定的 IP 地址。同时每个 Load Generator 上运行大量的虚拟用户，这样就造成了大量的用户使用同一 IP 同时访问一个网站的情况，这种情况和实际运行的情况不符，并且有一些网站会根据用户 IP 来分配资源，这些网站会限制同一个 IP 的登录、使用等。为了更加真实的模拟实际情况，LoadRunner 允许运行的虚拟用户使用不同的 IP 访问统一网站，这种技术称为“IP 欺骗”。

启用该选项后，场景中运行的虚拟用户将模拟从不同的 IP 地址发送请求。

注意：*IP Spoofer 在连接 Load Generators 之前启用。要使用 IP 欺骗，各个 Load Generator 机器必须使用固定的 IP，不能使用动态 IP(即 DHCP)。*

使用 IP Spoofer 的步骤如下。

① 使用 IP Wizard。在【开始】菜单【所有程序】中，执行【LoadRunner】→【Tools】→【IP Wizard】命令，弹出如图 7.49 所示 IP Wizard 设置窗口。

注意：*运行 IP Wizard 程序的机器必须使用固定的 IP，不能使用动态 IP。*

第一次运行 IP Wizard 需要选择第一项【Create new settings】，如果以前运行过，可以选择第二项【Load previous settings from file】，选择保存好的文件；第三项【Restore original set】用于使用 IP 欺骗进行测试完成后，释放 IP 的过程，由于该机会占用大量的 IP 资源，可能会导致其他机器没有 IP 可用的尴尬局面，使用该项，可以恢复到原来的状况。

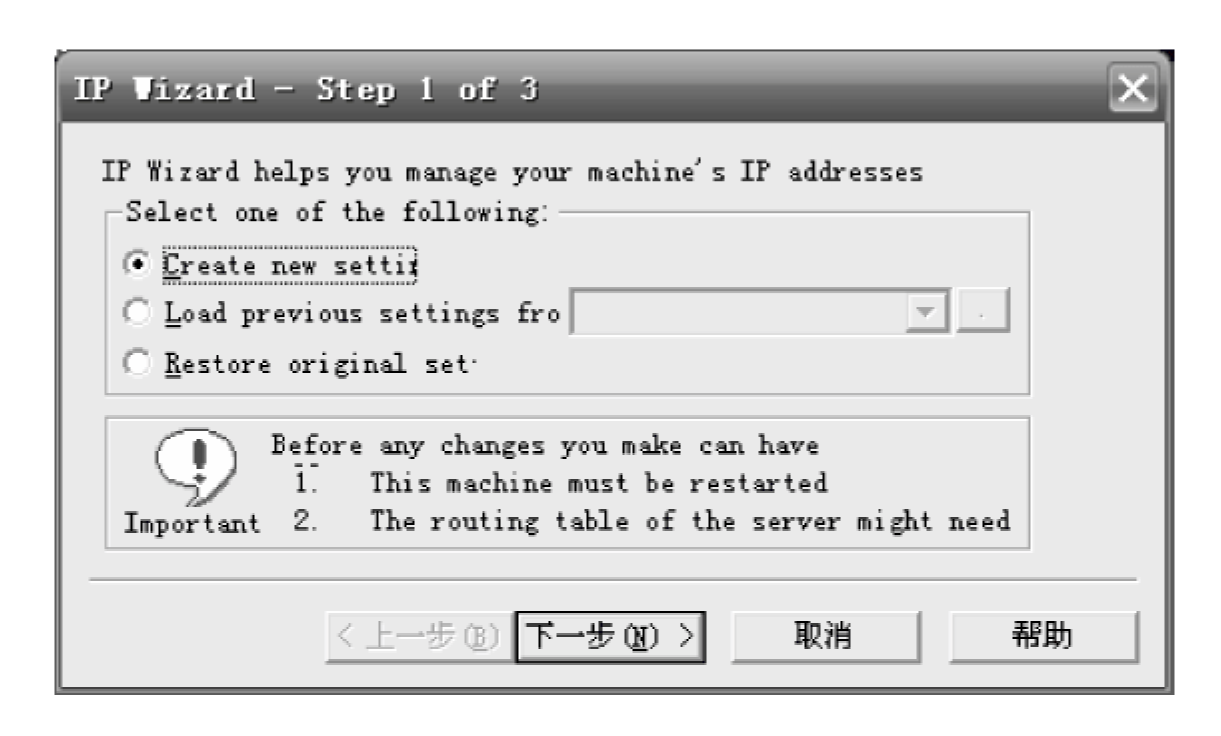


图 7.49 IP Wizard 设置窗口(1)

这里选择第一项单击【下一步】按钮，出现 IP Wizard 的第二个窗口，如图 7.50 所示。

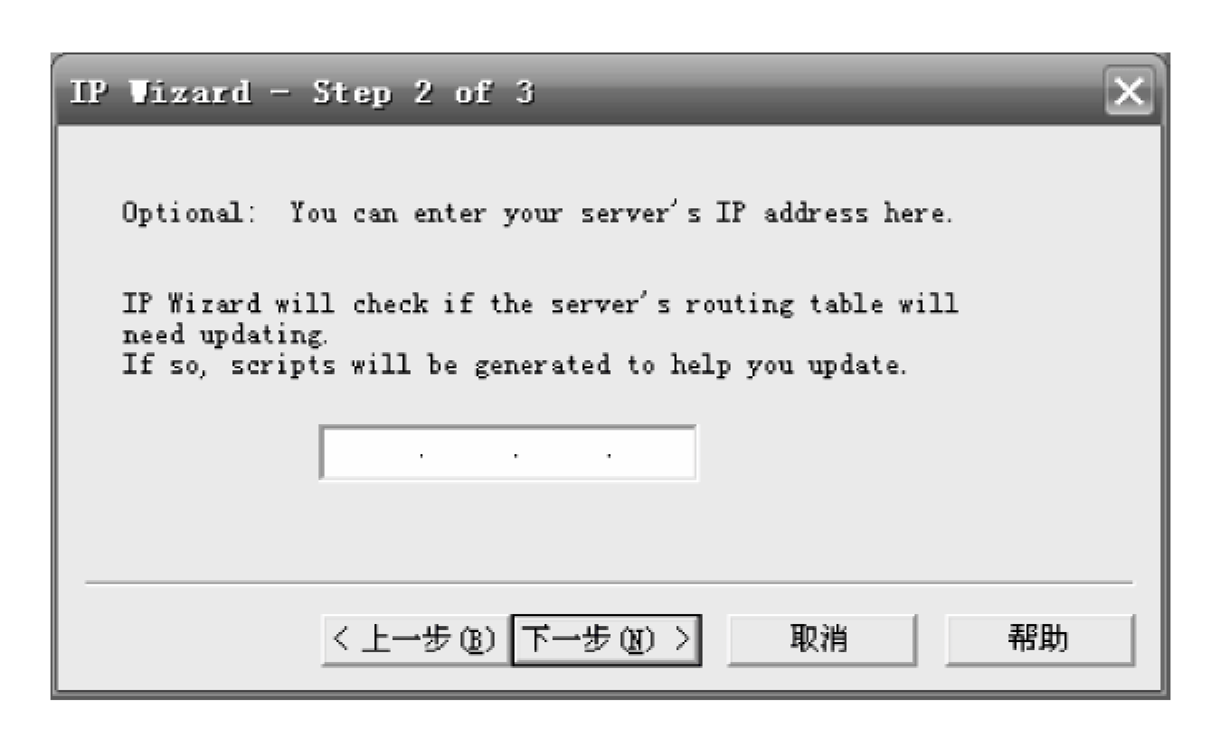


图 7.50 IP Wizard 设置窗口(2)

这里输入 Web Server 的 IP 地址，然后单击【下一步】按钮，出现向导的第三个窗口，如图 7.51 所示。

图 7.51 IP Wizard 设置窗口(3)

单击【Add】按钮进入 Add 设置菜单，如图 7.52 所示。

在【From IP】文本框中输入要使用 IP 范围的第一个 IP 值，然后在【Number to】文本框中输入一个数字，表示 IP 范围的值；假如第一个 IP 为 192.168.6.100，范围大小为 100，那么 IP Wizard 将会使用 192.168.6.N(100≤N<200)，当然这个范围内已经使用的 IP 地址除外，否则会引起 IP 冲突。【Submask】采用默认情况即可，一般局域网内采用 Class C。单击【OK】按钮，IP Wizard 开始检查该范围内没有使用的 IP，并把没有使用的 IP 添加到本机

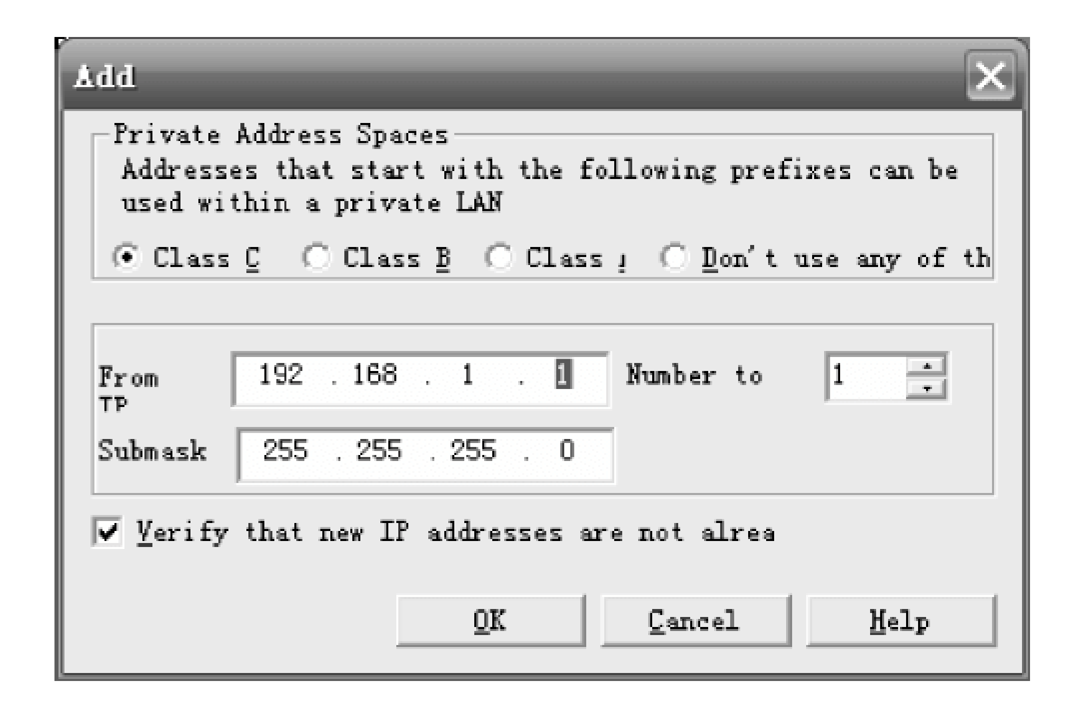


图 7.52　Add 窗口

的 IP 窗口中。到最后一个窗口，单击【Finish】按钮，使用 IP Wizard 后，需要重新启动计算机。

② 在 Controller 的场景中，启用 IP Spoofer 即可，如图 7.53 所示。

在菜单栏执行【Scenario】→【Enable IP Spoofer】命令，使该项被选中，同时状态栏中会显示使用“IP Spoofer”的标志。

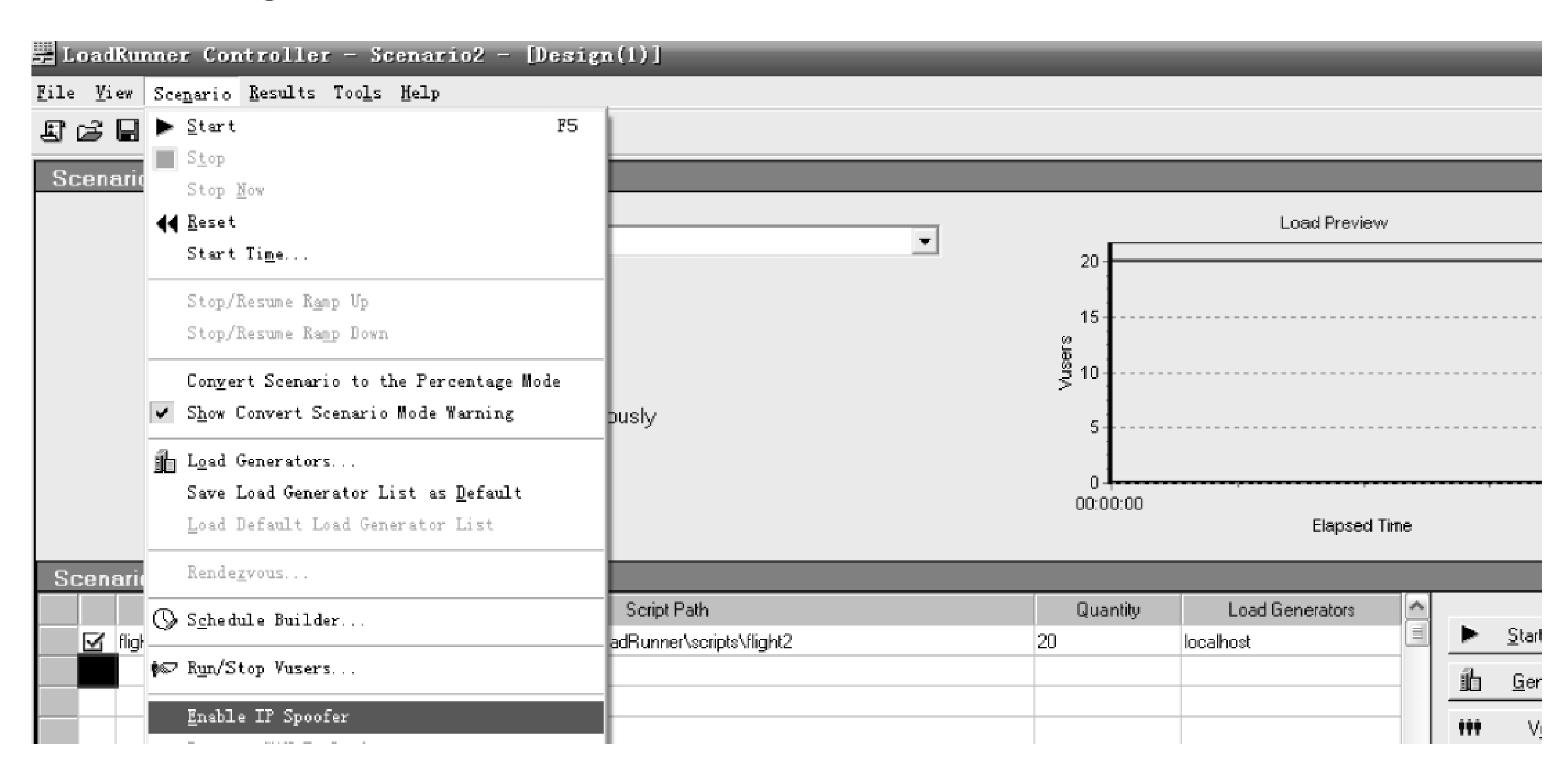


图 7.53　在 Controller 场景中，启用 IP Spoofer

(3) 优化 Controller 和 Load Generators 计算机

如果控制机(Controller machine)和 Load Generators 计算机运行的都是 Windows 2000，那么下面两个简单的技巧可以提高性能。

- 在 Load Generators 计算机上，依次进入【控制面板】→【系统】→【高级】标签页，单击【性能选项】按钮，选择优化【后台服务】选项，这样可以提高性能，从而可以在每个 Load Generators 上运行更多的虚拟用户。
- 在 Controller 计算机上，按照以上的步骤，进入【性能选项】窗口，不过这里选择优化【应用程序】。

7.3.4　运行测试

一切配置妥当，开始运行测试。

7.3.5 监视场景

在运行过程中，LoadRunner 可以监视它所支持的以下服务器的资源。

（1）System Resource：包括 Windows 平台，UNIX 平台等。

（2）Web Server：包括 Apache、IIS、SUN 的 iplanet 等。

（3）Application Server：包括 Weblogic、WebSphere 等。

（4）Database Server：包括 DB2，Oracle，SQL Server，Sybase 等。

（5）Java：EJB，J2EE 等，需要一个 ejbdetector.jar 文件。

监视场景需要在 Run 视图中通过添加性能计数器来实现，如图 7.54 所示。

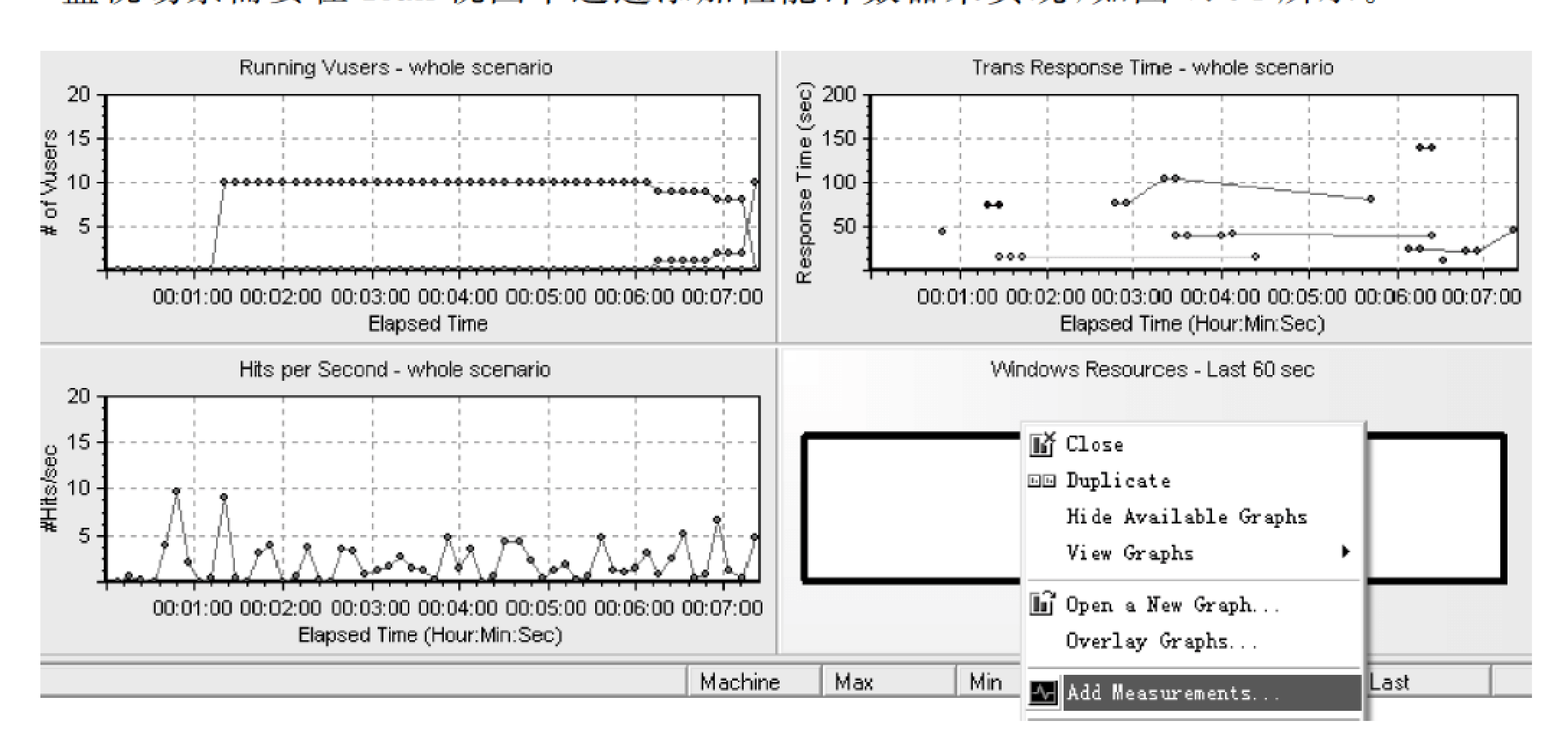


图 7.54 Run 视图中添加性能计数器

然后，出现添加计数器的对话框，如图 7.55 所示。

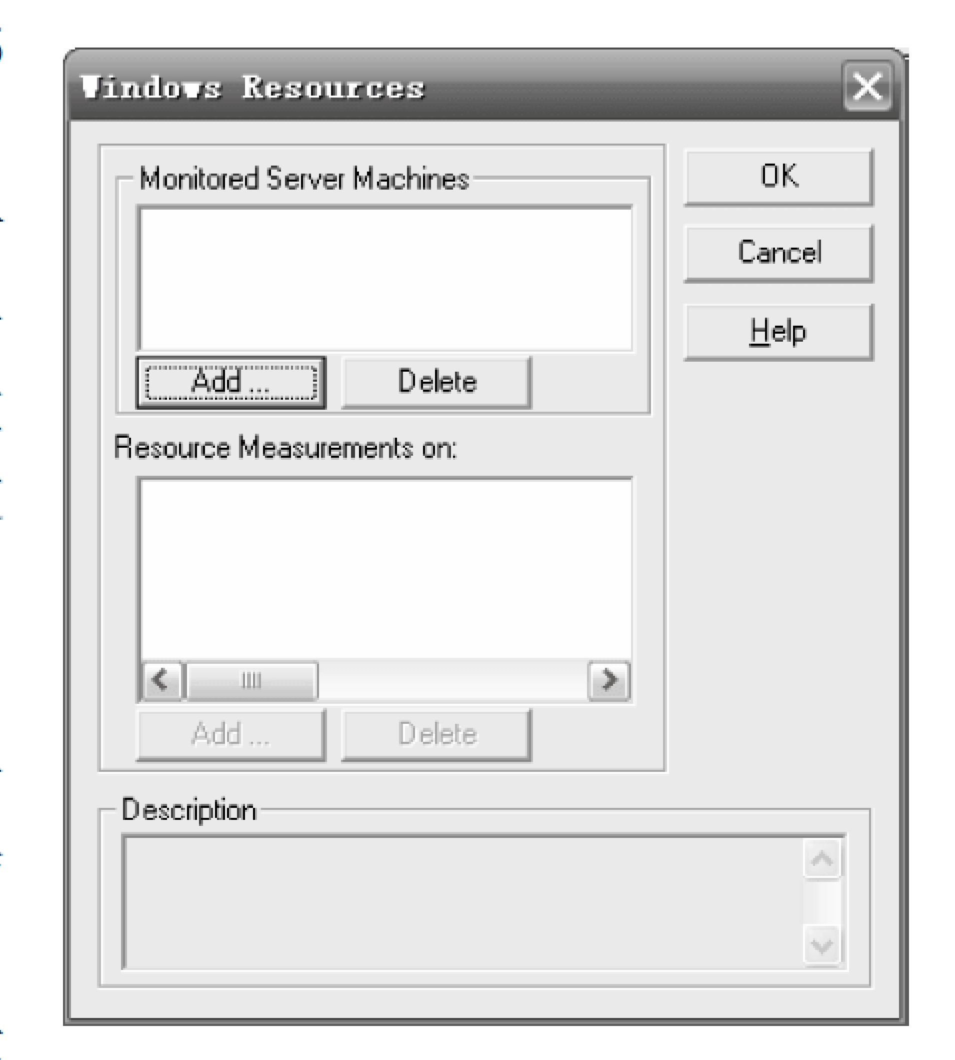


图 7.55 计数器对话框

其他的操作就和控制面板“性能”中添加性能计数器的操作一样，这里不再详说。本章主要说明一下各个系统计数器的含义（数据库的计数器不做重点，因为数据库各个版本之间差异比较大，请参考使用的数据库系统的帮助）。

1. Memory 相关

内存是第一个监视对象，确定系统瓶颈的第一个步骤就是排除内存问题。内存短缺的问题可能会引起各种各样的问题。

内存问题主要检查应用程序是否存在内存泄漏。如果发生了内存泄漏，Process\Private Bytes 计数器和 Process\Working Set 计数器的值往往会升高，同时 Available Bytes 的值会降低。内存泄漏应该通过一个长时间的，用来研究分析当

所有内存都耗尽时，应用程序反应情况的测试来检验。

2. Processor 相关

判断应用程序是否存在处理器瓶颈的方法是：如果 Processor Queue Length 显示的队列长度保持不变(≥2)并且处理器的利用率%Processor Time 超过 90%，那么很有可能存在处理器瓶颈。

如果发现 Processor Queue Length 显示的队列长度超过 2，而处理器的利用率却一直很低，那么或许更应该去解决处理器阻塞问题，这里处理器一般不是瓶颈。

如果系统由于应用程序代码效率低下或者系统结构设计有缺陷而导致大量的上下文切换(Context Switches/sec 显示的上下文切换次数比较大)，那么就会占用大量的系统资源。如果系统的吞吐量降低并且 CPU 的使用率很高，并且此现象发生时切换水平在 15 000 以上，那么意味着上下文切换次数过高。

同时还可以比较 Context Switches/sec 和 Privileged Time 来判断上下文切换是否过量。如果后者的值超过 40%，且上下文切换的速率也很高，那么应该检查为什么会产生这样高的上下文切换。

3. 磁盘相关

判断磁盘瓶颈的方法是通过以下公式来计算：

每磁盘的 I/O 数=[读次数+(4×写次数)]/磁盘个数

如果计算出的每磁盘的 I/O 数大于磁盘的处理能力，那么磁盘存在瓶颈。

4. Network Delay

如果要监视的两台计算机在同一个局域网络内，建议不要使用 Network Delay Monitor。因为在同一局域网内，Network Delay 会非常的小，网络监视器会有足够的时间在每秒钟内发送成百上千的请求，这样会导致源计算机(source machine)的 CPU 和内存超负荷工作。

默认情况下“Enable display of network nodes by DNS names”选择是没有选中的，因为选中它会明显的降低该监视器的速度。

7.3.6 利用 Analysis 分析结果

场景运行结束后，需要使用 Analysis 组件分析结果。Analysis 组件可以在【开始程序】菜单中启动，也可以在 Controller 中启动。

这里只是按照常规的方法进行简单介绍。

注意：这里介绍的分析方法只适用于 Web 测试。

1. 分析事务的响应时间

(1) 看【Transaction Performance Summary】图，确认哪个事务的响应时间比较大，超出了规定的标准。

(2) 看【Average Transaction Response Time】图，观察各事务在整个场景运行中每一秒的情况。

(3) 定位问题需要分解事务，分析该页面上每一个元素的性能，选择要分解的事务曲线，然后右击选择【Web Page Breakdown for login】项。

2. 分解页面

通过分解页面可以得到：比较大的响应时间到底是页面的哪个组件引起的，问题出在服务器上还是网络传输上。

这里为了解说各个时间(比如：DNS 解析时间、连接时间、接受时间等)简单说一下浏览器从发送一个请求到最后显示的全过程。



(1) 浏览器向 Web Server 发送请求，一般情况下，该请求首先发送到 DNS Server 把 DNS 名字解析成 IP 地址。解析的过程的时间就是 DNS Resolution 。这个度量时间可以确定 DNS 服务器或者 DNS 服务器的配置是否有问题。如果 DNS Server 运行情况比较好，该值会比较小。

(2) 解析出 Web Server 的 IP 地址后，请求被送到 Web Server，然后浏览器和 Web Server 之间需要建立一个初始化连接，建立该连接的过程就是 Connection 。这个度量时间可以简单的判断网络情况，也可以判断 Web Server 是否能够响应这个请求。如果正常，该值会比较小。

(3) 建立连接后，从 Web Server 发出第一个数据包，经过网络传输到客户端，浏览器成功接受到第一字节的时间就是 First Buffer 。这个度量时间不仅可以表示 WebServer 的延迟时间，还可以表示出网络的反应时间。

(4) 从浏览器接受到第一个字节起，直到成功收到最后一个字节，下载完成止，这段时间就是 Receive 。这个度量时间可以判断网络的质量(可以用 size/time 比来计算接受速率)。

其他的时间还有 SSL Handshaking(SSL 握手协议，用到该协议的页面比较少)；ClientTime(请求在客户端浏览器延迟的时间，可能是由于客户端浏览器的 think time 或者客户端其他方面引起的延迟)；Error Time(从发送了一个 HTTP 请求，到 Web Server 发送回一个 HTTP 错误信息，需要的时间)。

3. 确定 Web Server 的问题

网站的性能问题可能是由多种因素引起的，其中大约有一半的性能问题最终归结到 Web Server、Web 应用程序和数据库服务器上。采用编程语言(ASP、JSP、ASP. NET 等)的网站非常依赖于数据库操作，这些都可能是引起性能问题的因素。最常见的数据库问题是效率比较低的索引设计，数据碎片太多，过时的统计表以及不完善的应用程序设计。

在 20%的压力测试中，发现 Web Server 和 Web 应用程序是性能的瓶颈。这些瓶颈主要是由于服务器配置不当和资源不足。比如，编程比较差的代码以及形成的 DLL 能够使用所有的计算机处理器资源，导致了 CPU 的瓶颈。同样，对内存的操作不当和管理不善也很容易造成内存的瓶颈，故建议在排除其他可能的因素外，首先检查 CPU 和物理内存。

4. 其他有用的功能——比较每次运行的结果

一般情况下，进行性能测试的步骤如下。

（1）首先进行一次性能测试，记录下结果，然后分析结果，提出改进的建议；

（2）开发人员根据建议对代码或者服务器的配置进行修改；

（3）测试人员在相同的条件下进行第二轮测试；

（4）测试人员对两轮测试结果比较，确定开发人员修改的结果是否有效。

那么在 Analysis 中怎样进行对两轮结果进行比较呢？

可以在菜单栏中执行【File】→【Cross With Result】命令，然后的在出现的对话框中单击【Add】按钮，通过文件所在路径选择需要比较的结果文件(lrr)，如图 7.56 所示。

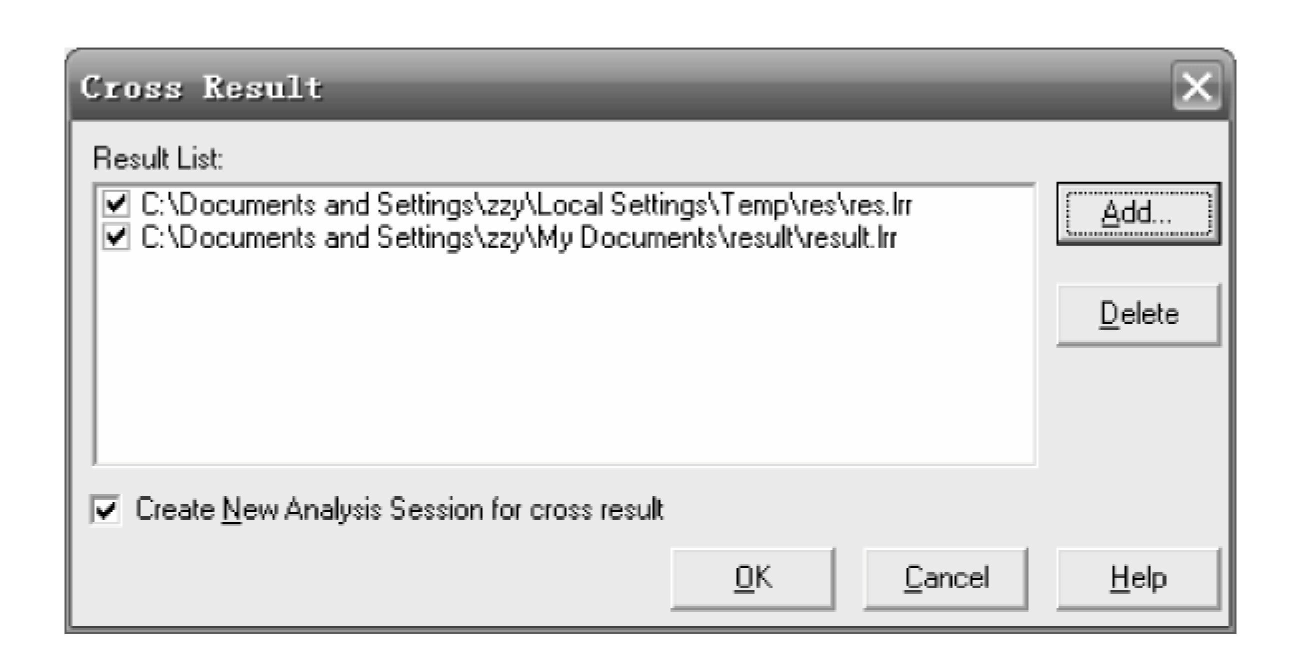


图 7.56　Cross Result 窗口

单击【OK】按钮即可看到比较的结果，如图 7.57 所示。

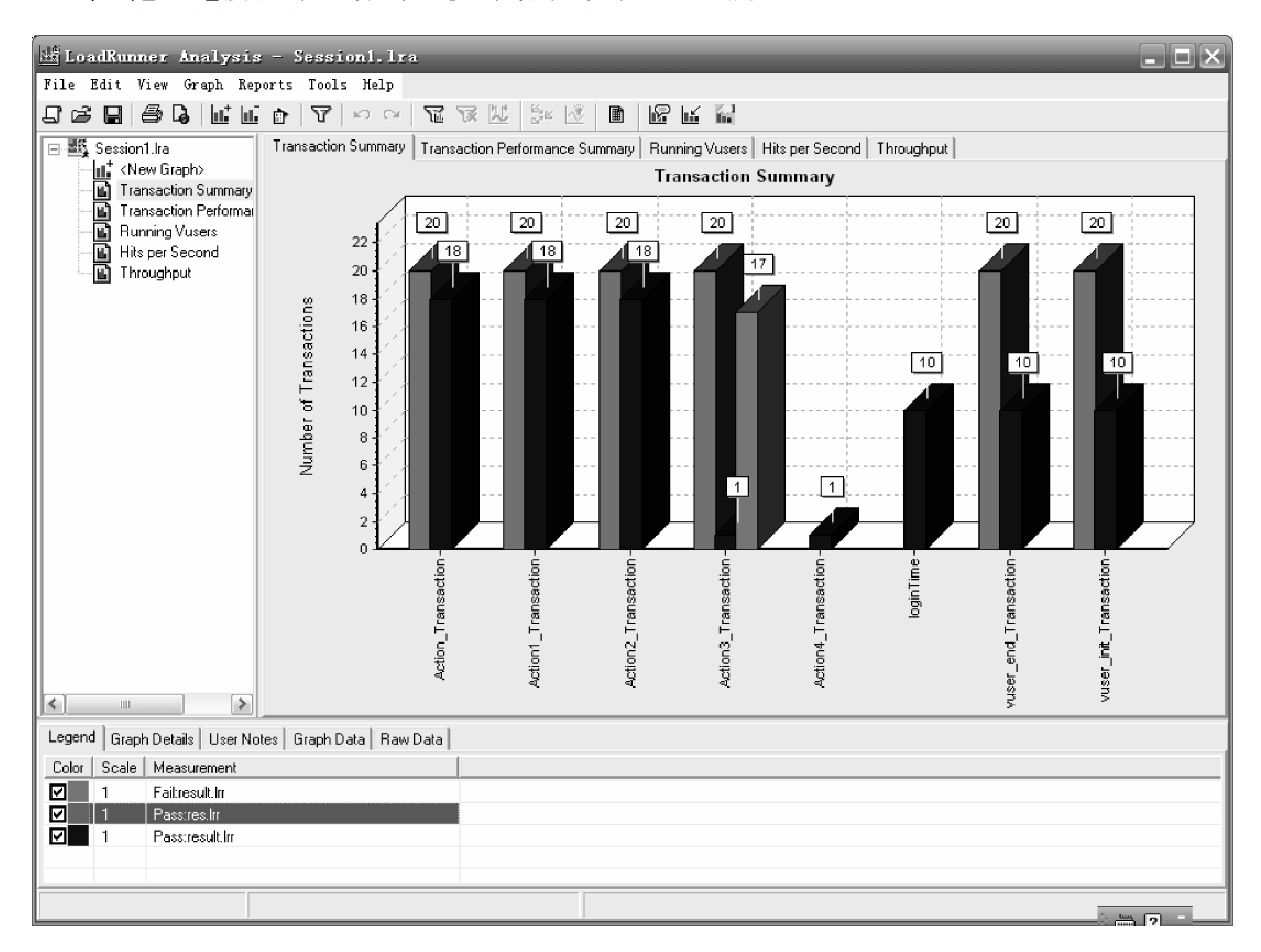


图 7.57　结果分析图

7.4 LoadRunner 测试实例

7.4.1 项目背景信息

近两年，随着网络的发展，视频网站如雨后春笋般出现。尤其一些热门的视频网站，拥有巨大的用户群体。如一些知名电视台的宽频网站，在社会发生热点新闻期间的并发用户访问数量会达到百万级以上。巨大的并发访问量对系统的性能提出了非常高的要求。

本案例探讨的是一个已经上线的视频网站遇到的性能问题，该系统的设计目标是每天150万的PV(页面浏览)量。在上线后，由于用户并发量较大，CPU的利用率经常达100%，导致Oracle数据库发生停止服务的现象。数据库不工作，网站运营人员就无法维护系统，甚至导致终端用户不能正常访问网站。显然，这类问题是不能容忍的。

在探讨性能测试工作之前，先简要介绍这个系统的体系结构，其功能点将在后面的内容中介绍，整个系统由下面两块组成。

视频发布系统：该系统是一个成型的产品，主要功能是建立一个运营平台，把视频发布到系统中，并为门户提供接口。本系统已经有多家成功应用的案例。

网站门户：对视频发布系统进行二次开发后实现的一个应用。借助视频发布平台提供的接口，门户实现了与视频发布系统所维护的运营平台之间的信息交互。网站的用户主要通过门户来欣赏视频。

下面探讨如何测试系统的性能问题。

7.4.2 测试执行与结果分析

对本系统而言，由于已经发现了问题，所以性能测试的目标非常明确，就是要找出导致数据库停止服务的原因。而分析这类问题时，很容易想到两个常见的原因：程序算法上的缺陷和数据库配置不正确。算法上的缺陷会导致CPU资源过度消耗，而配置不正确也会引起数据库系统运行异常。因此，性能测试设计和实施将围绕这两个目标来进行。

下面详细介绍实验室的性能测试过程。

1. 测试用例设计

由于问题出在数据库上，因此测试用例应该针对数据库来进行。数据库的测试通常会分为下面3个步骤。

首先，把数据库的操作分为Insert、Update、Delete、Select共4种，分别隔离进行测试，并定位哪种操作容易引起问题；

其次，把用户的日常操作模拟出来，也就是把用户对数据库的操作组合起来进行测试；

最后，做一些疲劳强度或大数据量的压力测试，以使问题快速重现。

确定了整体方案后，接下来就要确定测试过程需要模拟哪些用户操作。分析整个系统的结构，可以看出视频发布系统出问题的可能性不大，因为这是一个成型的产品。因此，问题更可能会出现在网站的门户或门户与视频发布系统的接口上。

经过进一步的分析，了解到网站门户用户访问量主要有以下3个页面。

视频首页：视频首页是导航页面，主要操作有查找热点视频或自己关注的视频、进入二

级分类页面、进入播放页面等。

收费播放页面：付费用户播放视频时进入的页面。

免费播放页面：用户播放免费视频时进入的页面。

这3个页面应该是重点测试的对象，用户日常操作组合测试、疲劳强度与大数据量测试都应该针对它们进行。确定了测试内容后，就可以开始性能测试的实施了。

2. 测试实施过程

不难看出，实验室测试是一个紧急的性能测试任务，因此没有时间进行正规的规划与设计，更多的是一种应变测试。为了保证系统的"正常"运行，环境设在了实验室里，配置如表7.3所示。

表7.3　测试硬件配置

服务器类型	硬 件 配 置	软 件 配 置
数据库服务器	服务器：两台 Dell 2850 CPU：Xeon 3.0GB×2 内存：2GB	操作系统：企业版 Windows 2003(SP1) 数据库：Oracle10g
应用服务器		操作系统：企业版 Windows 2003(SP1) Web Server：IIS6.0

由于硬件环境的差异，实验室里的调优只能作为上线后的参考。实验室测试适合找出由软件自身缺陷引起的性能问题，较容易发现一些算法方面的缺陷。

(1) 常规并发用户测试

由于实验室与用户现场的硬件环境差别较大，因此应该先在实验室中进行"预测试"，看看系统在实验室中的性能表现，为后面的测试提供参考。在本案例中，先选择50个并发用户来观察系统的性能表现，压力持续时间为30分钟。

注意：*读者碰到类似项目时可以根据系统自身的特点来选择并发用户数量，这里的50只是个参考。*

经过测试后，利用Analysis进行分析，得到了如图7.58所示的测试结果摘要。从图中可以看出，视频首页(Index页面)不能访问，收费播放页面(Play访问)的平均访问时间为181.834秒。

Statistics Summary

Maximum Running Vusers:	50	
Total Throughput (bytes):	139,332,618	
Average Throughput (bytes/second):	77,921	
Total Hits:	29,476	
Average Hits per Second:	16.357	View HTTP Responses Summary

Transaction Summary

Transactions: Total Passed: 109 Total Failed: 593 Total Stopped: 98　　Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action_Transaction	0	0	0	0	0	0	201	49
IndexMM	0	0	0	0	0	0	201	49
PlayMM	175.185	181.834	189.472	4.779	189.472	9	191	0
vuser_end_Transaction	0	0	0.002	0	0.002	50	0	0
vuser_init_Transaction	0	0.038	0.382	0.069	0.085	50	0	0

图7.58　50个并发用户的预测试结果

由此不难得出结论：这是一个过于缓慢的系统，需要进行调整后才能正常开展测试工作。

通常情况下，对这类系统进行调整首先应从系统的参数配置或硬件配置入手，然后分析软件自身的原因。这样做的原因是参数配置不正确的错误很容易纠正，也是常见的性能问题，而分析软件本身则是后面测试工作的主要内容。因此，在查看了 Oracle 的服务器配置后，立刻发现了一个参数配置问题：Oracle 数据库的运行模式是"专有服务器模式"，而"共享服务器模式"才是更适合大规模并发的模式。

关于 Oracle 服务器的运行模式

在专用服务器模式下，用户连接所需要的全部资源在 PGA 中进行分配。该内存区为指定私有连接，其他进程不能访问。专用连接采用一对一的连接方式，能很快地响应用户的请求。但是，当用户连接过多时，由于要对每一个用户连接分配资源，因此，连接个数受硬件的限制比较大。为了克服这种情况，Oracle 提供了共享服务器运行模式，即用一个服务器的进程响应多个用户连接。只要实例一启动，就分配指定数量的服务器进程，所有用户的连接以排队的方式由分配器指定给服务器进程，其他的进程排队等待。只要用户的请求执行完，就会马上断开连接，分配器会把空闲的服务器进程分配给其他排队的进程。

因此，首先要把 Oracle 的运行模式调整为"共享服务器模式"再进行测试。

分析"Summary"的意义

从上面的内容可以看出，"Summary"的分析结果决定了是否继续进行后面的工作。在本案例中，通过"Summary"看出系统性能非常令人不满意，因此没有必要进行深入分析。正确的做法是立刻采取调优措施，否则测试工作将无法正常开展下去。

(2) 独立场景测试

下面是 Oracle 调整为"共享服务器模式"后的测试实施过程。为了更好地对问题定位，测试只针对播放页面来进行。如图 7.59 所示是 800 个用户并发、压力持续 30 分钟的测试结果综述。

Statistics Summary

Maximum Running Vusers:	800	
Total Throughput (bytes):	66,145,330,967	
Average Throughput (bytes/second):	36,706,632	
Total Hits:	15,677,791	
Average Hits per Second:	8,700.217	View HTTP Responses Summary

Transaction Summary

Transactions: Total Passed: 804,127 Total Failed: 174,004 Total Stopped: 600 Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action_Transaction	0	1.354	66.116	1.904	1.381	802,527	174,004	600
vuser_end_Transaction	0	0	0	0	0	800	0	0
vuser_init_Transaction	0	0	0.025	0.001	0	800	0	0

图 7.59　共享模式下 800 个用户的并发测试结果

从图 7.59 中可以看出,“Action”事务即打开播放页面的平均时间是 1.354 秒,这是非常好的结果。但是应该注意到:半小时内有超过 17 万个播放页面不能正常打开,同时计算出为“打开播放页面”事务的通过率为 82%。82%的事务通过率标志着后续的性能测试任务十分艰巨,而半小时内超过 17 万个播放页面不能正常打开,是任何视频网站都不能接受的。

为了让问题更好地暴露出来,再进行了一次更长时间的场景测试:压力持续时间增加到三倍,即 1.5 个小时;并发用户增加到了 900 个。测试结果如图 7.60 所示。

Statistics Summary

Maximum Running Vusers:	900
Total Throughput (bytes):	2.0977067251E11
Average Throughput (bytes/second):	45,534,047
Total Hits:	68,547,846
Average Hits per Second:	10,771.189

View HTTP Responses Summary

Transaction Summary

Transactions: Total Passed: 3,545,018 Total Failed: 623,251 Total Stopped: 800   Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action_Transaction	0.012	1.438	103.906	1.986	1.534	3,543,218	623,251	800
vuser_end_Transaction	0	0	0.016	0	0	900	0	0
vuser_init_Transaction	0	0	0.016	0.001	0	900	0	0

图 7.60　共享模式下 900 个用户的并发测试结果

与图 7.59 的测试结果相比,得到如下结论:“打开播放页面”事务的平均响应时间稍稍变大,这是用户数量变大的正常反应;事务通过率 85%与 82%相比没有太大变化。稍稍提高的通过率很可能是由于测试时间较长,打开的部分页面存在于服务器缓存中造成的。

这时,打开 Oracle 的管理工具,借助 Oracle 提供的工具进行分析。结果发现了 5 个问题,如图 7.61 所示。

影响 (%)	查找结果	建议案
93.7	发现 SQL 语句消耗了大量数据库时间。	2 SQL Tuning
90	缓冲区高速缓存不够大,从而导致大量的附加读 I/O。	1 DB Configuration
89.43	发现个别数据库段造成了大量的用户 I/O 等待。	2 Segment Tuning
55.24	等待类别 "其它" 消耗了大量数据库时间。	
55.24	等待事件 "class slave wait" (在等待类 "Other" 中) 消耗了大量数据库时间。	2 Application Analysis

图 7.61　数据库分析的结果

打开图 7.61 中的“发现 SQL 语句消耗了大量数据库时间”链接进行查看,发现 Oracle 找出了 3 个 SQL 语句的问题,而这 3 个语句恰恰是播放页面频繁使用的语句。这样找到了程序本身的一个原因:SQL 语句消耗了大量的数据库时间,其他问题极有可能是语句不合理引起的。主要推理如下:当一些反复执行的 SQL 语句效率过低时,首先会造成高速缓存不够用,随之引起较大的 I/O;而频繁的 I/O 势必会消耗大量的 CPU。因此整个系统的瓶颈极有可能是这 3 个语句引起的。

测试过程中棘手的性能问题

这两次测试结果还有一个奇怪的现象:一方面事务响应时间较快,另一方面却有大量的事务没有响应。仅根据目前的测试结果还看不出直接原因。但这也很可能是前面的 3 个 SQL 语句引起的:因为这种现象很像用户直接收到不能访问的通知,所以不再等待服务器的结果反馈,在客户端的表现就是页面无法打开。

这种现象很有可能是耗资源的 SQL 语句瞬间霸占了全部 CPU 资源，导致数据库拒绝服务。当调整了 SQL 语句再次进行测试时，这种现象消失，证明推理是正确的。

再借助 Analysis 打开图 7.62 所示的“事务平均响应时间图”和图 7.63 所示的“建立第一个缓冲的时间分解图”，可以得出以下结论。

① 事务平均响应时间稳定，说明本次测试的性能问题主要在程序自身。如果是服务器存在问题，则长时间的压力测试会导致响应时间越来越长。

② “建立第一个缓冲的时间”主要消耗在服务器上，说明对用户请求的处理方式不合理。

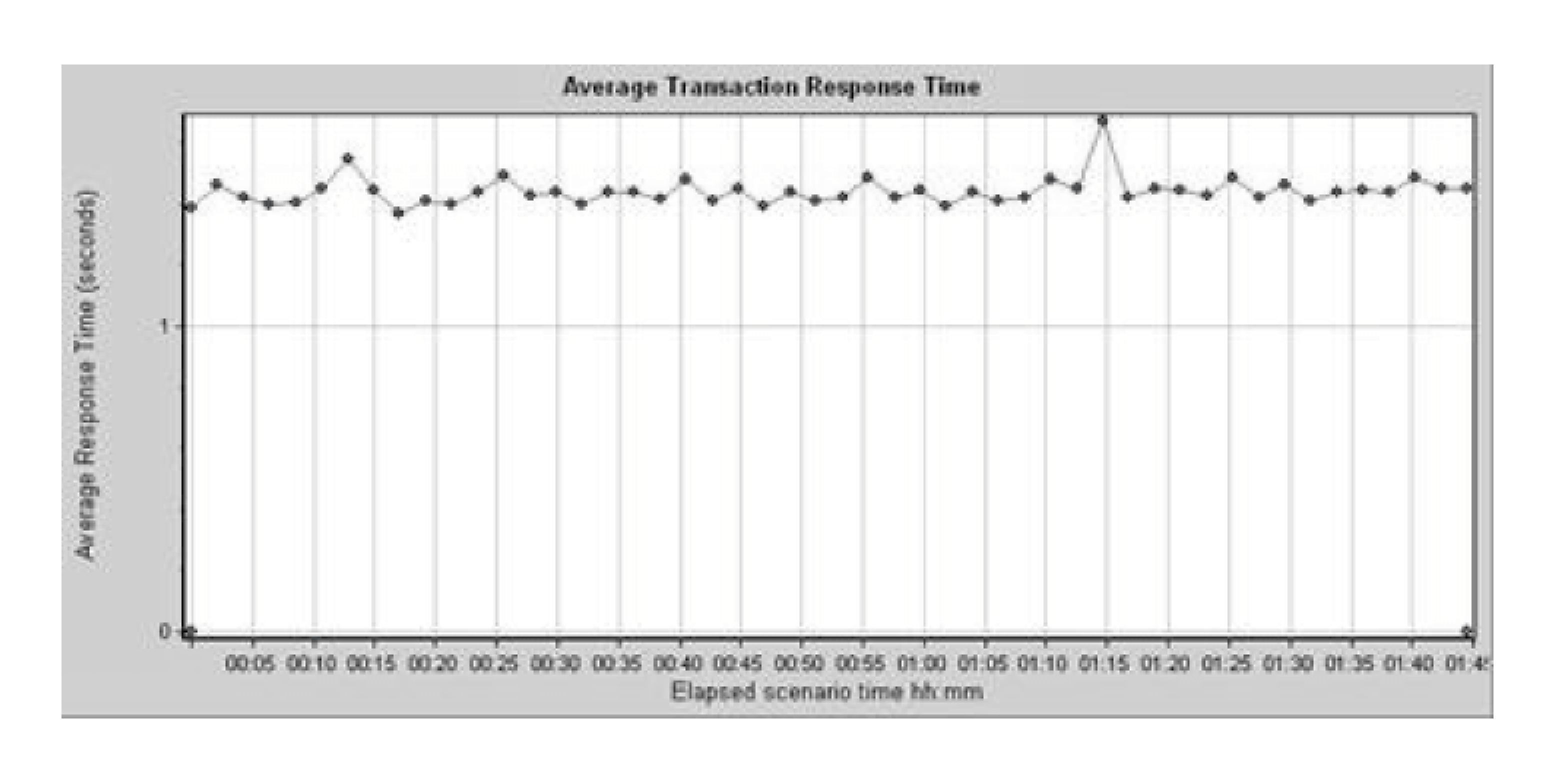


图 7.62　事务平均响应时间图

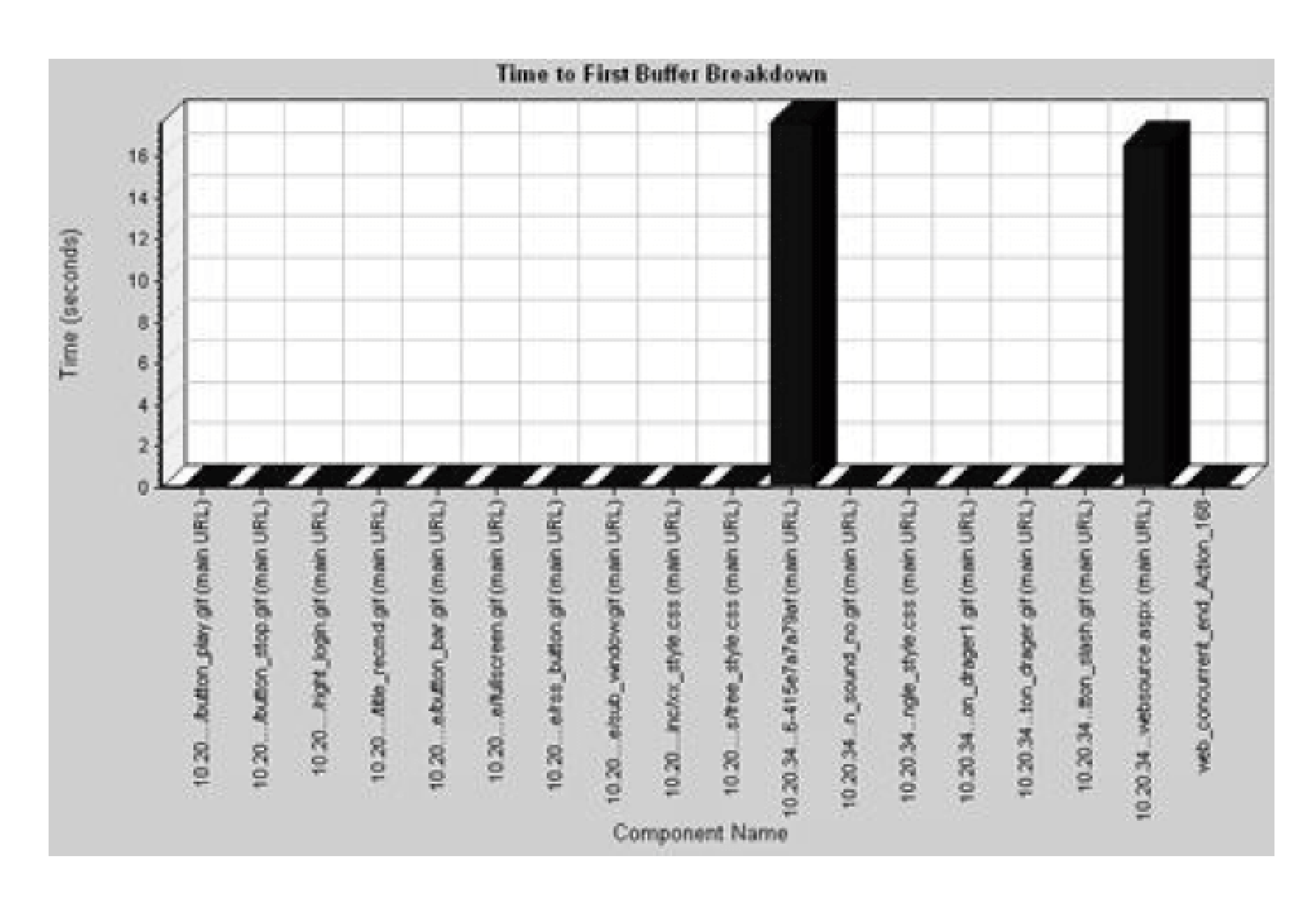
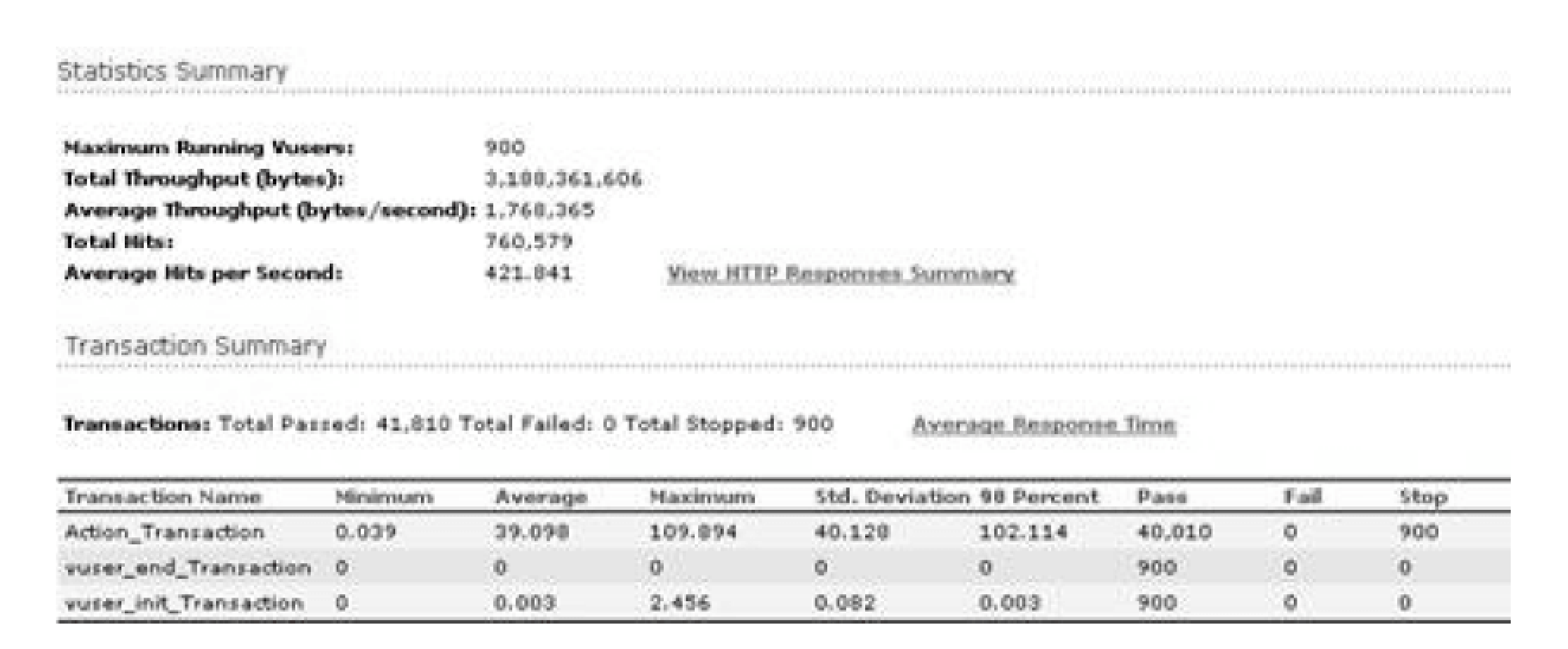


图 7.63　建立第一个缓冲的时间分解图

因此，下一步应该先对 SQL 语句进行优化，然后再对系统进行测试。

（3）SQL 语句调整后的测试

开发人员优化了 SQL 语句后，并对 900 个用户进行并发、持续 1.5 小时的压力测试，测试结果如图 7.64 所示。从图中可以看出，调整后系统的性能非常稳定，所有的用户均可以成功打开“视频播放页面”。



Statistics Summary

Maximum Running Vusers: 900
Total Throughput (bytes): 3,188,361,606
Average Throughput (bytes/second): 1,768,365
Total Hits: 760,579
Average Hits per Second: 421.841 View HTTP Responses Summary

Transaction Summary

Transactions: Total Passed: 41,810 Total Failed: 0 Total Stopped: 900 Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	98 Percent	Pass	Fail	Stop
Action_Transaction	0.039	39.098	109.894	40.120	102.114	40,010	0	900
vuser_end_Transaction	0	0	0	0	0	900	0	0
vuser_init_Transaction	0	0.003	2.456	0.082	0.003	900	0	0

图 7.64　SQL 语句调整后的测试

再借助 Analysis 打开如图 7.65 所示的事务平均响应时间图，可以看到整个测试过程“打开播放页面”的平均响应时间成平滑水平线，系统的性能非常稳定。

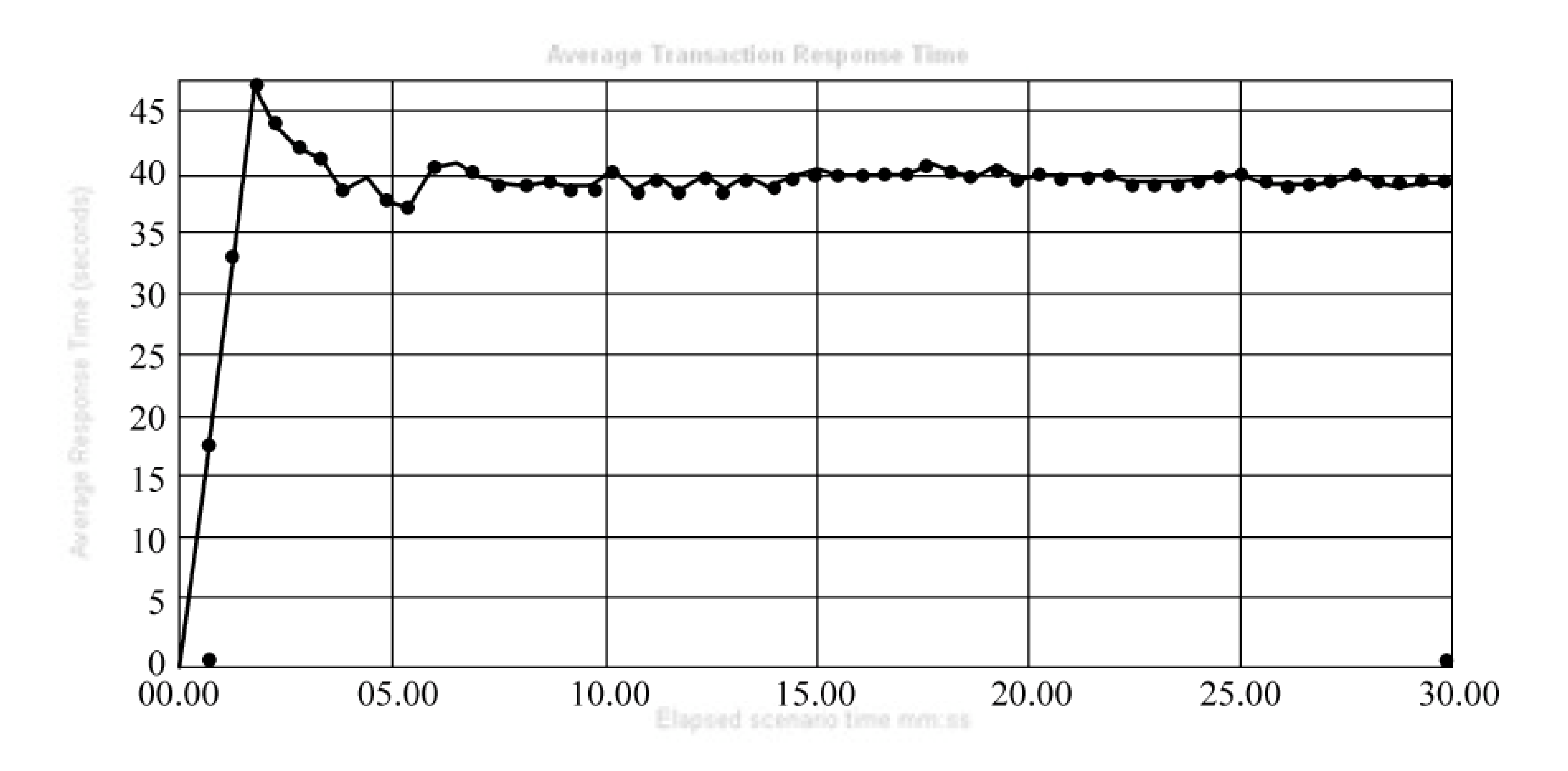


图 7.65　调整后的事务平均响应时间图

由此可以得出结论：调整 SQL 语句的策略是正确的。

与图 7.59 和图 7.60 所示的测试结果相比，有个细节在分析的时候应该注意到，即事务平均响应时间变长了。前面两次测试结果的事务平均响应时间是 1 秒，而本次则是 39 秒。那么，哪次的测试结果更合理呢?

根据常理，本次测试用的是普通的 PC 服务器，900 个并发用户用 1 秒的响应时间显然不合理，39 秒才符合实际情况。1 秒的事务平均响应时间只是一种假象，是系统存在性能问题造成的。对于“播放页面”的性能，在图 7.64 中有稳定表现，可以认为测试过关!

到目前为止，本次性能测试的思路是正确的，可以推广到其他功能的独立或组合性能测试中。

3. 性能调优方案

最后，对整个上线系统提出了以下调优方案。

系统配置方面的调优方案如下。

(1) 把 Oracle 的运行模式调成“共享服务器模式”。

(2) 增加了分配给 Oracle 的内存,把内存调整成系统内存的 55%。

(3) 增加共享池(SHARED_POOL)和缓冲区高速缓存(DB_CACHE)的大小。

(4) 对数据库表和查询相关的字段建立索引。

应用程序方面的调优方案如下。

(1) 用优化后的程序来替换原有程序。

(2) 采用页面缓存技术来提高用户访问速度,同时减缓对数据库的压力。

线上的性能测试验证:按照调优方案对线上系统进行调整后,接下来的工作就是和客户一起验证系统的综合性能表现。线上测试与实验室测试不同,不能影响用户的正常使用——不能产生垃圾数据,更不能把服务器压垮。

在前面的项目背景中,提到过本系统的设计目标是满足每天 150 万的 PV(页面浏览量),因此,线上测试主要是为了评估服务器能够处理的浏览量。同时,为了保证服务器的稳定运行,在真实线上环境只模拟了 500 个并发用户。

7.4.3 测试结果

下面是对线上系统首页的测试结果摘要,主要测试了动态页面无缓存、静态页面缓存这两种方式。

1. 页面无缓存、500 用户并发

测试场景持续执行时间:6 分钟

运行的最大用户数:1000 个

测试内容:打开任意视频进行播放

事务平均响应时间如图 7.66 所示。

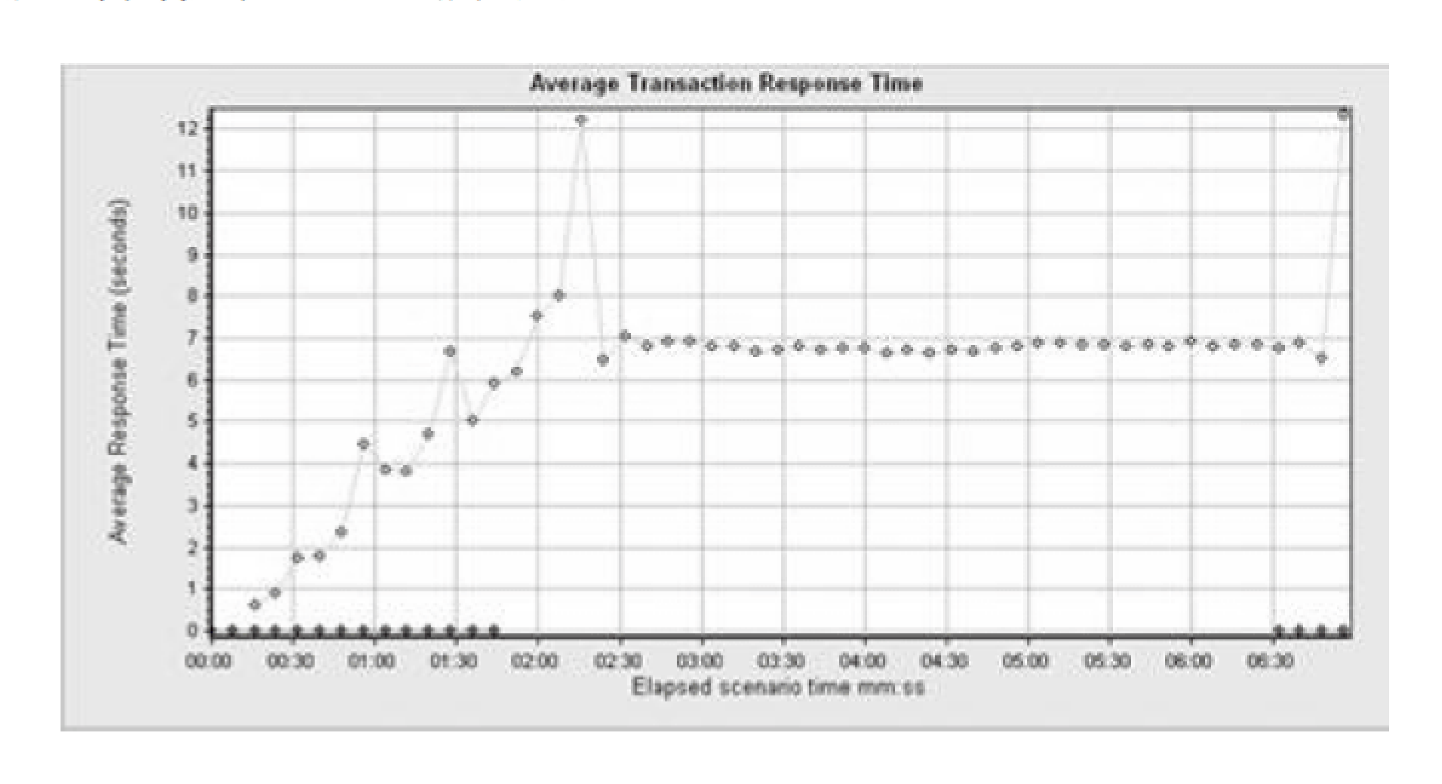


图 7.66 事务平均响应时间

事务响应时间的详细情况如表 7.4 所示。

表 7.4 事务响应时间(秒)

最小值	平均值	最大值
0.622	6.247	12.338

CPU 利用率如图 7.67 所示。

其中服务器 CPU 利用率(%)的详细情况如表 7.5 所示。

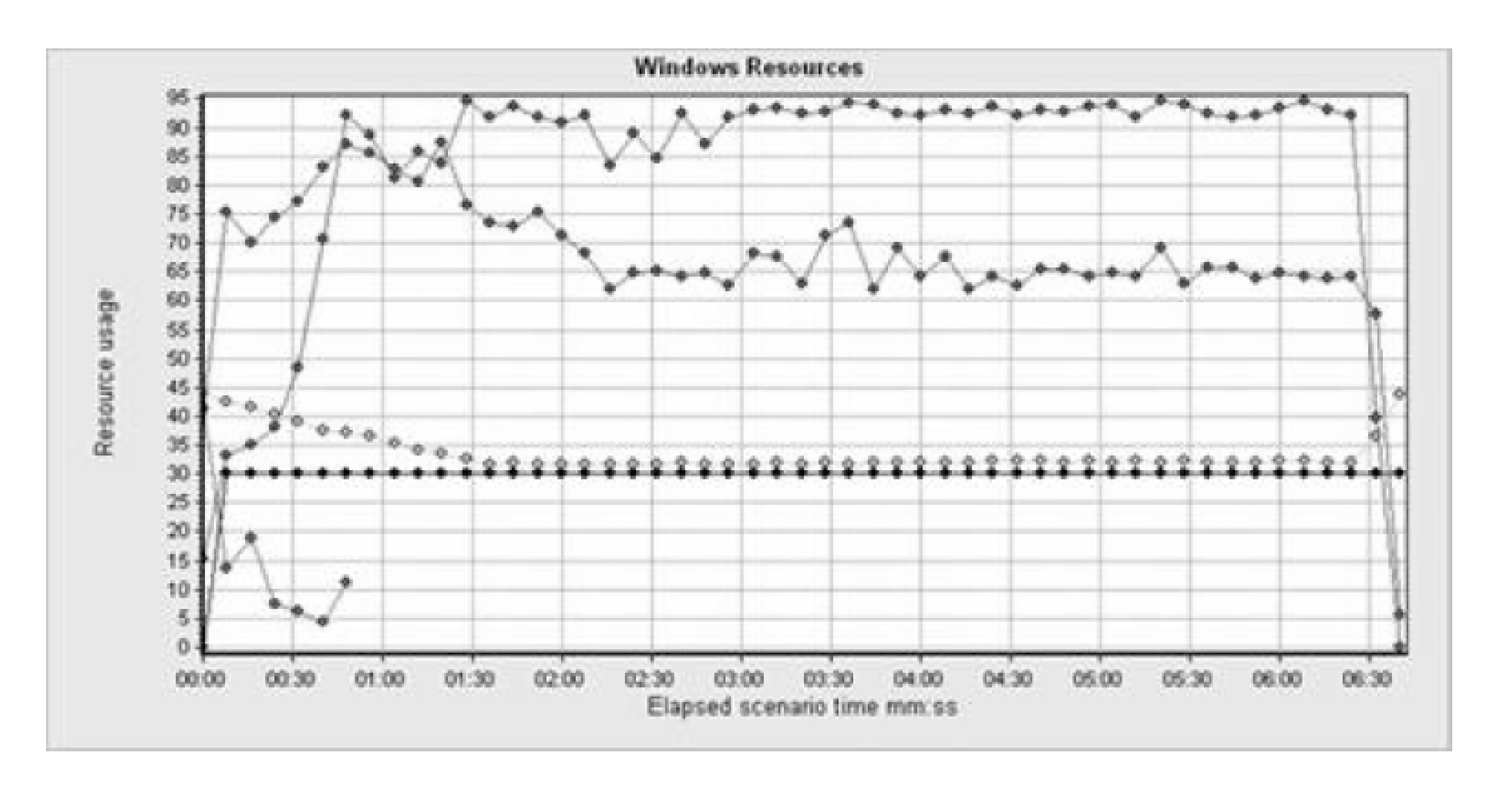


图 7.67　CPU 利用率

表 7.5　服务器 CPU 利用率

服务器	最小值	平均值	最大值
Oracle 服务器	0.0	82.723	95.703
Web 服务器	4.688	67.543	95.833

每秒播放页面下载数如图 7.68 所示。

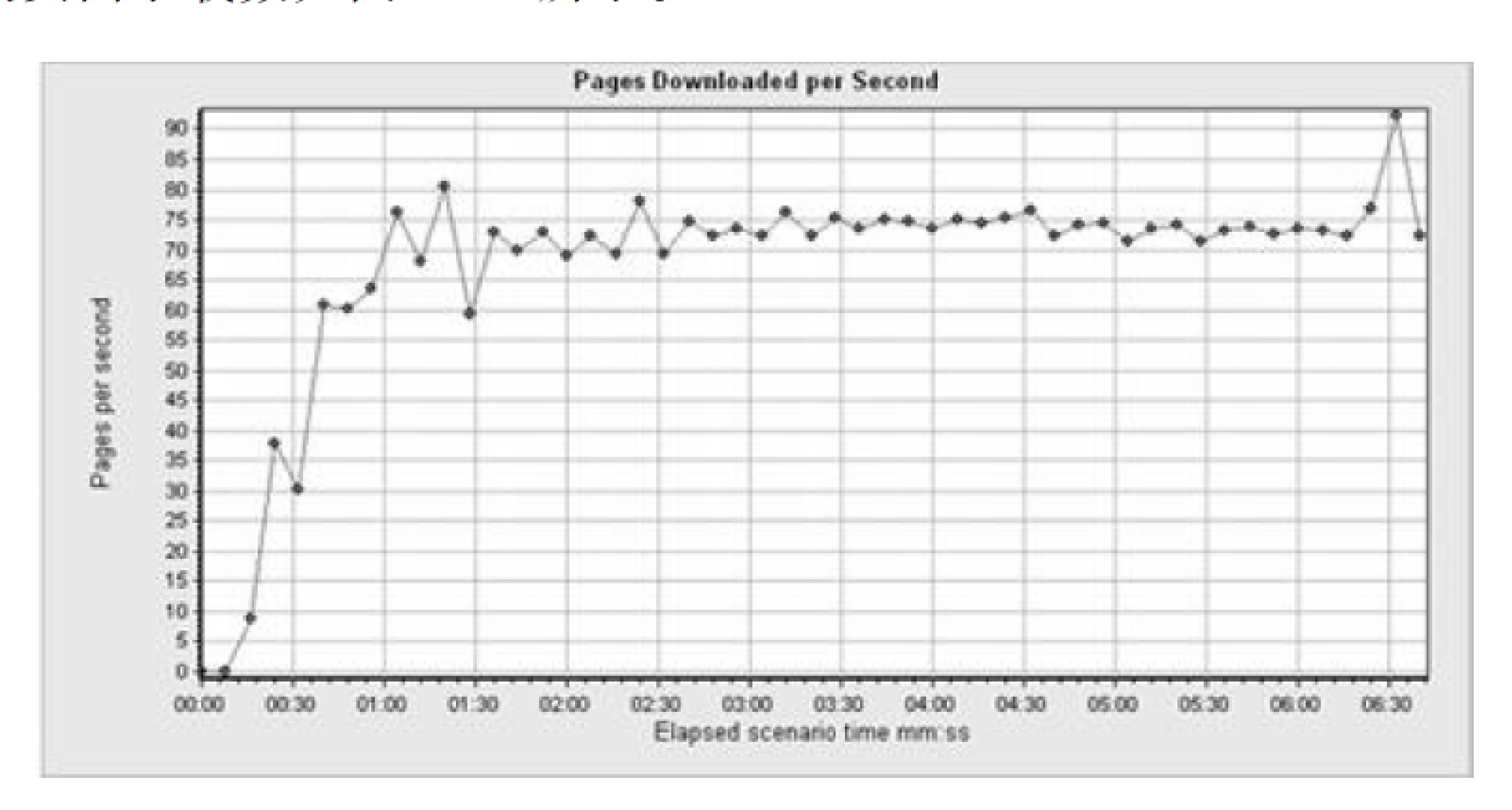


图 7.68　每秒页面下载数

每秒播放页面下载数的详细情况如表 7.6 所示。

表 7.6　每秒播放页面下载数

最小值	平均值	最大值
0.0	67.126	92.25

本次测试结论：Oracle 服务器 CPU 的平均利用率为 82.723%，说明数据库系统稳定运行。Web 服务器的 CPU 平均利用率是 67.543%。如果按一天 8 小时计算，一台服务器每天的 PV 均值为 67.126×3600×8≈193 万个，足可以支撑 150 万个 PV。

2. 采用静态页面缓存方式、500 用户并发

测试场景持续执行时间：6 分钟

运行的最大用户数：1000 个
测试内容：打开任意视频进行播放
测试过程中的事务平均响应时间如图 7.69 所示。

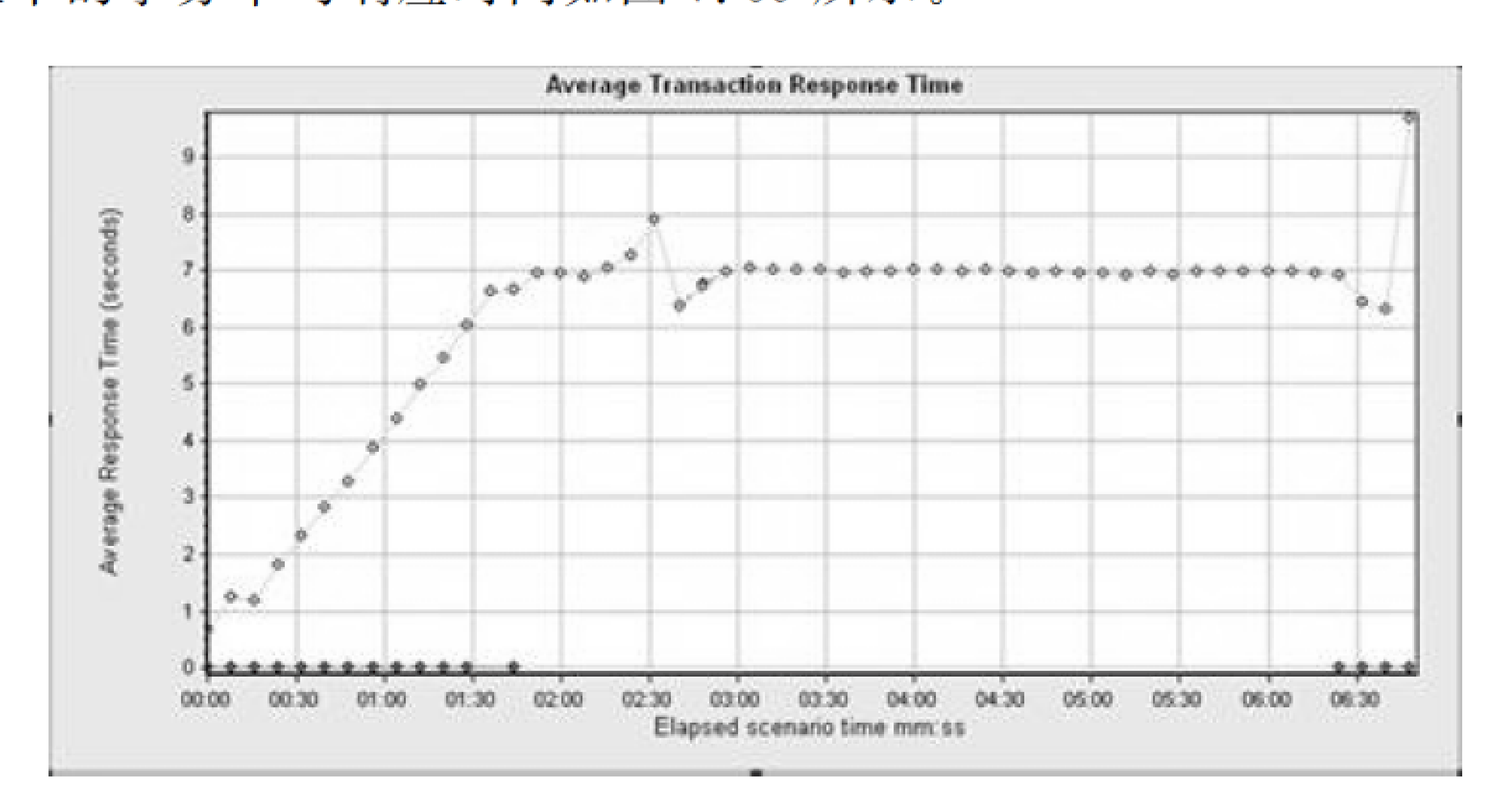


图 7.69　事务平均响应时间

事务平均响应时间的详细情况如表 7.7 所示。

表 7.7　事务平均响应时间

最小值	平均值	最大值
0.682	6.124	9.682

测试过程中 CPU 利用率如图 7.70 所示。

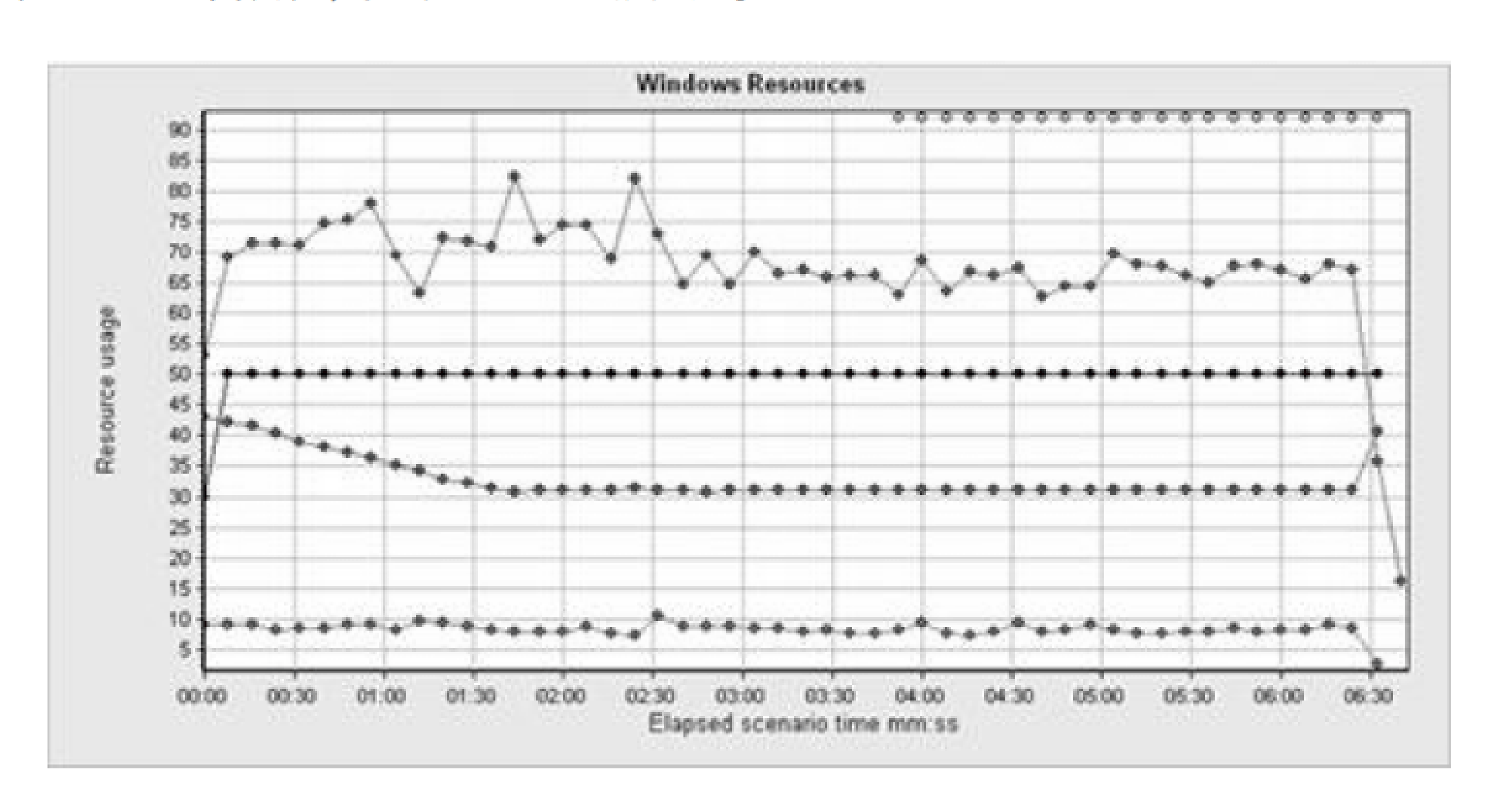


图 7.70　CPU 利用率

其中服务器 CPU 利用率(%)的详细情况如表 7.8 所示。

表 7.8　CPU 利用率(%)测试结果数据

服务器	最小值	平均值	最大值
Oracle 服务器	0.0	8.267	15.445
Web 服务器	3.125	67.327	89.119

测试过程中每秒播放页面下载数如图 7.71 所示。
每秒播放页面下载数的详细情况如表 7.9 所示。

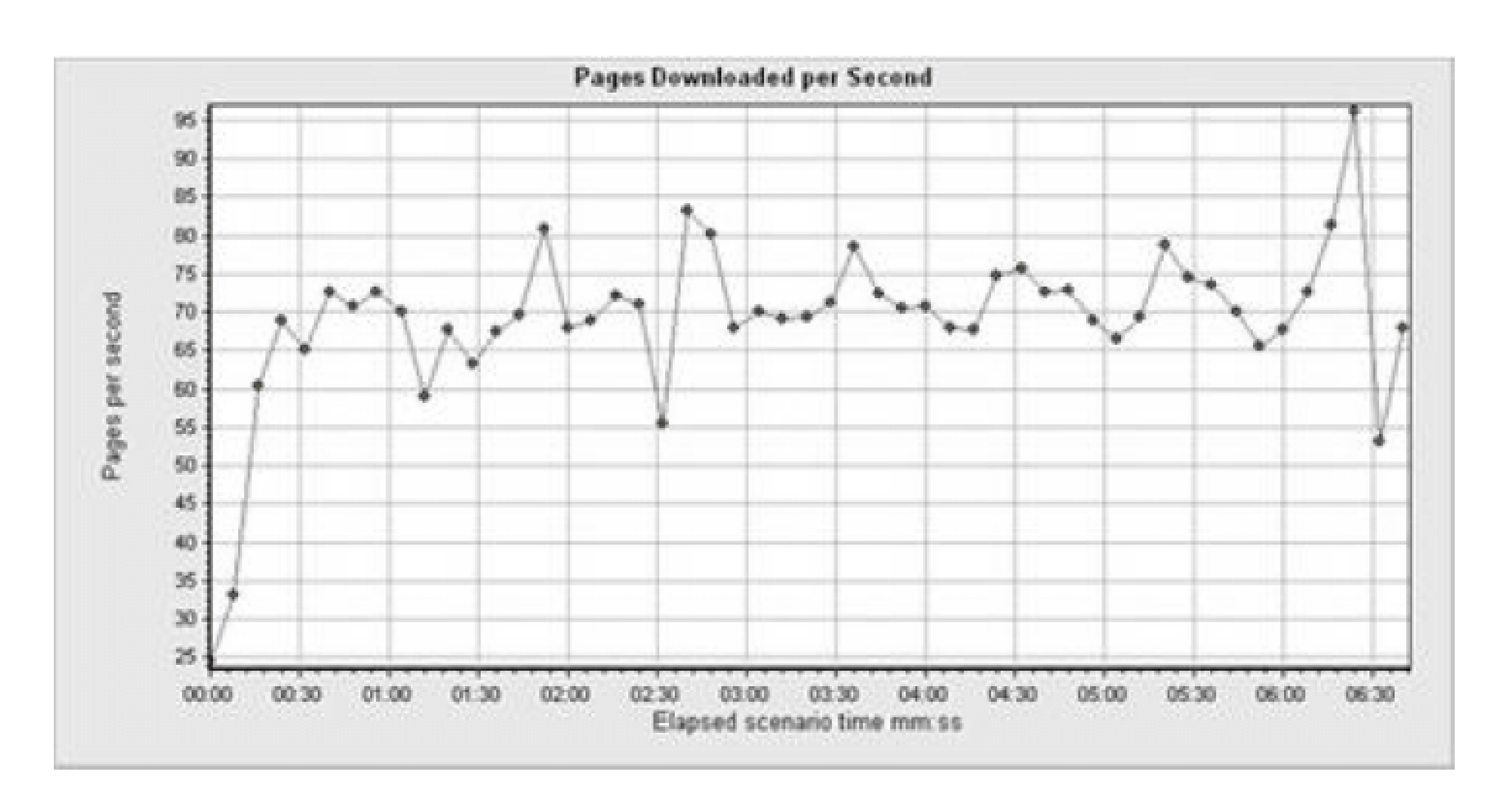


图 7.71　每秒页面下载数

表 7.9　播放页面下载结果数据

最小值	平均值	最大值
24.375	69.043	96.125

本次测试结论：Oracle 服务器 CPU 的平均利用率非常低，为 8.267%，这说明静态页面缓存技术大大节省了对数据库的资源消耗，系统更加稳定运行。Web 服务器的 CPU 平均利用率是 67.323%。如果按一天 8 小时计算，一台服务器每天的 PV 均值为 69.043×3600×8≈199 万个，足可以支撑 150 万个 PV。

在满足系统性能测试目标的前提下，Oracle 数据库 CPU 的利用率不足 10%，标志着本次性能测试工作圆满完成。

7.4.4　案例总结

通过本项目的性能分析，应更加体会到性能分析是一个相当复杂的过程，仅靠 Analysis 是远远不够的。在实际工作中，往往会借助各种系统工具以及各方面的综合知识找出系统的瓶颈。例如在本项目中，Oracle 自带的管理工具起到了至关重要的作用，本案例恰恰是借助它发现了引起系统瓶颈的 SQL 语句，借助这个突破口逐步解决了其他性能问题。

当进行性能分析与调优时，应该先从容易的地方入手。这样做的原因是可以先排除常见的、容易引起瓶颈的问题，避免各种因素混杂在一起不容易对问题进行定位。例如，本案例就是先从 Oracle 参数配置入手，逐步深入到应用程序内部。

在实际项目中，还应该要细心地查看 Analysis 的各种分析报表，不能让任何一个性能问题漏掉。在抓住了系统存在问题后，就可以深入地分析。

练　习　题

1. 试用 LoadRunner 所给的示例，根据自己的理解设计测试，制定负载测试计划、开发负载测试脚本、创建运行场景、运行测试以及利用 Analysis 分析结果。

2. 如何利用 LoadRunner 判断 HTTP 服务器的返回状态。

3. 一个公司的系统上线以后，用户分布在各个不同的地区，而且接入系统的方式和带宽也不同，这种情况下进行性能测试，如何更加真实的模拟用户行为？用 LoadRunner 可以做到吗？

4. 在 Web 应用下，模拟 10 个用户并发进行数据的添加，结果每次执行全部成功，但是数据却不是 10 条，每次数据不一样，但是都比 10 小。这种情况产生的原因是什么？

5. 在 LoadRunner 下如何让多个场景轮流执行？

6. 请解释 LoadRunner 下最大并发用户数、业务操作响应时间、服务器资源监控指标的含义与用途。

第 8 章 JUnit

8.1 JUnit 概述

JUnit 是一个开源的 java 测试框架，它是 Xuint 测试体系架构的一种实现。JUnit 最初由 Erich Gamma 和 Kent Beck 所开发。在 JUnit 单元测试框架的设计时，设定了三个总体目标，第一个是简化测试的编写，这种简化包括测试框架的学习和实际测试单元的编写；第二个是使测试单元保持持久性；第三个则是可以利用既有的测试来编写相关的测试。

8.2 JUnit 的安装

8.2.1 命令行安装

JUnit 是以 jar 文件的形式发布的，其中包括了所有必须的类。安装 JUnit，所需要做的一切工作就是把 jar 文件放到编译器能够找到的地方。

如果不使用 IDE，而是从命令行直接调用 JDK，那么必须让 CLASSPATH 包含 JUnit 的 jar 包所在的路径。

(1) 在 Linux 或者其他类 UNIX 的系统上，只需要把 jar 文件的路径加入到 CLASSPATH 环境变量之中。例如：假设 jar 文件位于/usr/java/packages/junit3.8.1/junit.jar。只需要运行类似这样的一条命令：

```
CLASSPATH = SCLASSPATH:/usr/java/packages/junit3.8.1/junit.jar
```

class path 中的每条路径都要用冒号隔开（“：”）。

通常会把这样的命令放入到 shell 的启动脚本之中(.bashrc 或者/etc/profile 或者类似的位置)，因此就不需要总是重复的修改 CLASSPATH 了。

(2) 在微软的 Windows 操作系统中，进行下面这个菜单路径：

```
Start
   └Settings
        └Control Panel
              └System
                   └Advance Tab
                         └Environment Variables...
```

如果有了修改已经存在的哪个 CLASSPATH 变量，或者添加一个名为 CLASSPATH 的环

境变量。假设 JUnit 的 jar 包位于 C:\java\junit3.8.1\junit.jar，需要把这些值输入哪个对话框中：

```
Variable:CLASSPATH
Variable Value:C:\java\junit3.8.1\junit.jar
```

如果在 class path 中有已经存在的条目，注意每个新加的 class path 都要用分号（“;”）隔开。

需要重新启动所有的 shell 窗口或者应用程序以使这些改动生效。

8.2.2 检查是否安装成功

要获知 JUnit 是否已经安装好了，试着编译一下包含下面这个 import 语句的源文件：

```
Import junit.framework.*;
```

如果这样成功了，那么编译器就能找到 JUnit 了，也就是说一切都准备妥当了。不要忘记测试代码需要在 JUnit 的 TestCase 基类继承而来。

8.3 使用 JUnit 编写测试

8.3.1 构建单元测试

在编写测试代码的时候，需要遵循一些命名习惯。如果有一个名为 createAccount 的被测试函数，那么第一个测试函数的名称也许就是 testCreateAccount。其中，方法 testCreateAccount 以恰当的参数调用 createAccount 并验证 createAccount 的行为是否和它宣称的一样。当然可以有许多测试方法来执行 createAccount。

如图 8.1 所示展示了两块代码之间的关系。

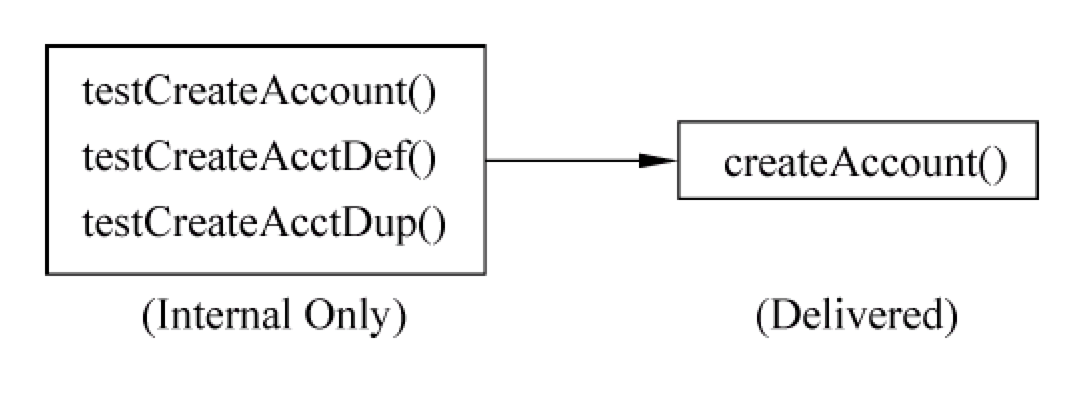


图 8.1　测试代码和产品代码

测试代码仅限于内部使用。客户或者最终用户永远都不会看到，更不会使用这些代码。因此，产品代码（最后要发布给客户或者放入产品中的代码）对测试代码是一无所知的，产品代码最后将撇下测试代码。

测试代码必须要做以下几件事情。

- 准备测试所需要的各种条件（创建所有必需的对象，分配必要的资源等）
- 调用要测试的方法
- 验证被测试方法的行为和期望是否一致
- 完成后清理各种资源

对于测试代码，也是用一般的方式编写和编译，这和项目中普通源码是一样的。测试代码可能偶尔会用到某些额外的程序库，但是除此之外，测试代码并没有任何特别之处，它们也只是普通代码而已。

当执行测试代码的时候，从来不直接运用产品代码。至少，并非像一个普通用户那样。

而是借助于测试代码,让它根据控制条件来执行产品代码。

通过在例子中使用 Java 语言展示 JUnit 的一些习惯,但是一般性的概念对于任何语言或者环境的任何测试框架都是一样的。下面就看看 JUnit 特有的一些函数和类。

8.3.2 JUnit 的各种断言

JUnit 提供了一些辅助函数,用于帮助确定某个被测试函数是否工作正常。通常而言,把所有这些函数统称为断言。可以确定:某条件是否为真;两个数据是否相等,或者不等,或者其他一些情况。在接下来的内容中,将逐个介绍 JUnit 提供的每一个断言(assert)方法。

下面每个方法都会记录是否失败了(断言为假)或者有错误了(遇到一个意料外的异常)的情况,并且通过 JUnit 的一些类来报告这些结果。对于命令行版本的 JUnit 而言,这意味着将会在命令行控制台上显示一些消息。对于 GUI 版本的 JUnit 而言,如果出现失败或者错误,将会显示一个红色条和一些用于对失败进行详细说明的辅助消息。

当一个失败或者错误出现的时候,当前测试方法的执行流程将会被中止,但是(位于同一个测试类中的)其他测试将会继续运行。

断言是单元测试最基本的组成部分,因此 JUnit 程序库提供了不同形式的多种断言。

1. assertEquals

```
assertEquals([String message],
             expected,
             actual)
```

这是使用得最多的断言形式。在上面的参数中,expected 是期望值(通常都是硬编码的),actual 是被测试代码实际产生的值,message 是一个可选的消息,如果提供的话,将会在发生错误的时候报告这个消息。当然,完全可以不提供这个 message 参数,而只提供 expected 和 actual 这两个值。

任何对象都可以拿来做相等性测试,适当的相等性判断方法会被用来做这样的比较。比如,可能会使用这个方法来比较两个字符串的内容是否相等。此外,对于原生类型(boolean、int、short 等)和 Object 类型也提供了不同的函数签名。值得注意的是使用原生数组的 equals 方法时,它并不是比较数组的内容,而只是比较数组引用本身,而这不是所期望的。

计算机并不能精确地表示所有的浮点数,通常都会有一些偏差。因此,如果想用断言来比较浮点数(在 Java 中,是类型为 float 或者 double 的数),则需要指定一个额外的误差参数。它表明需要多接近才能认为两数"相等"。对于商业程序而言,只要精确到小数点的后 4 位或者后 5 位就足够了。对于进行科学计算的程序而言,则可能需要更高的精度。

```
assertEquals([String message],
                     expected,
                     actual,
                     tolerance)
```

例如,下面的断言将会检查实际的计算结果是否等于 3.33,但是该检查只精确到小数

点的后两位。

```
assertEquals("Should be 3 1/3",3.33,10.0/3.0,0.01);
```

2. assertNull

```
assertNull([String message],java.lang.Object object)
assertNotNull([String message],java.lang.Object object)
```

验证一个给定的对象是否为 null(或者为非 null),如果答案为否,则将会失败。Message 参数是可选的。

3. assertSame

```
assertSame([String message],expected,actual)
```

验证 expected 参数和 actual 参数所引用的是否为同一个对象,如果不是的话,将会失败。message 参数是可选的。

```
assertNotSame([String message],expected,actual)
```

验证 expected 参数和 actual 参数所引用的是否为不同的对象,如果是相同的话,将会失败。Message 参数是可选的。

4. assertTrue

```
assertTrue([String message],Boolean condition)
```

验证给定的二元条件是否为真,如果为假的话,将会失败。Message 参数是可选的。

如果发现测试代码像下面这样:

```
AssertTrue(true);
```

那么该好好设计一下代码了。对于这种写法,除非是被用于确认某个分支,或者异常逻辑才有可能是正确的选择;否则的话,很可能是一个糟糕的主意。显然,在一页代码中看到只在该页的末尾出现许多 assertTrue(true)语句是不对的(也就是说,只是为了确认代码能够运行到末尾,没有中途死掉,并以为它就必然工作正常了)。

测试条件除了为真之外,也可以为假,如:

```
assertFalse([String message],Boolean condition)
```

上面代码用于验证给定的二元条件是否为假;如果不是的话(为真),该测试将会失败,message 参数是可选的。

5. Fail

```
Fail([String message])
```

上面的断言将会使测试立即失败,其中 message 参数是可选的。这种断言通常被用于标记某个不应该被到达的分支(例如,在一个预期发生的异常之后)。

6. 使用断言

一般而言,一个测试方法包含有多个断言,因为需要验证该方法的多个方面以及内在的多种联系。当一个断言失败的时候,该测试方法将会被中止,从而导致该方法中余下的断言

无法执行,此时只能在继续测试之前先修复这个失败的测试。依此类推,不断地修复一个又一个的测试,沿着这条路径慢慢前进。

期望所有的测试在任何时候都能通过。在实践中,这意味着当引入一个 bug 的时候,只有一到两个测试会失败。在这种情况下,把问题分离出来将会相当容易。

当有测试失败的时候,无论如何都不能给原有代码再添加新的特性。此时应该尽快地修复这个错误,直到让所有的测试都能顺利通过。

为了遵循上面的这种原则,需要一种能够运行所有测试或者一组测试,或某个特殊子系统的辅助方法。

8.3.3 JUnit 框架

到目前为止,只是介绍了断言方法本身。显然,不能只是简单地把断言方法写到源文件里面,而需要一个框架。

下面是一段简单的测试代码,它展示了开始使用的该框架的最小要求。

```
import junint.framework. * ;
public class Testsimple extends TestCase {
 public TestSimple (String name){
  super(name);
}
public void testAdd(){
  assertEquals(2,1 + 1);
}
}
```

尽管上面的代码非常清楚,但还是看看这段代码的每一部分。

首先,第 1 行的 import 声明引入了必须的 JUnit 类。

接下来,在第 3 行定义了一个类,每个包含测试都必须如所示那样由 TestCase 继承而来。基类 TestCase 提供了所需的大部分的单元测试功能,包括所有在前面讲述过的断言方法。

基类需要一个以 String 为参数的构造函数,因而必须调用 super 以传递这么一个名字。此时不知道这个名字是什么,因而仅仅让构造函数接受 String 为参数并把这个参数在第 5 行传递上去。

最后,测试类包含了名为 test 的方法。在上面这个例子中,在第 9 行写了一个名为 testAdd 的方法。而所有以 test 开头的方法都会被 JUnit 自动运行,还可以通过定义 suite 方法指定特殊的函数来运行,后面会做更多讲述。

在上面的例子中,展示了一个测试,它只有一个测试方法,而这个测试方法中又仅有一个断言。当然,在测试方法中,是可以写多个断言的,如:

```
public void testAdds(){
 assertEquals(2,1 + 14);
 assertEquals(4,2 + 2);
 assertEquals( - 8, - 12 + 4);
}
```

在此，一个测试方法里面使用了 3 个 assertEqual 断言。

8.3.4 JUnit 测试的组成

正如之前所看到的一样，一个测试类包含一些测试方法；每个方法包含一个或者多个断言语句。但是测试类也能调用其他测试类：单独的类、包甚至一个完整的系统。

这种魔力可以通过创建 test suite 来取得。任何测试类都能包含一个名为 suite 的静态方法，如：

```
public static Test suite();
```

可以提供 suite()方法来返回任何想要的测试集合(没有 siute()方法 JUnit 会自动运行所有的 test 方法)。但是可能需要手工添加特殊的测试，包括其他 suite。

例如，假设已经有了类似于在 TestClassOne 类中看到过的那样普通的一套测试，如：

```
import junit.framework. * ;
public class TestClassOne extends TestCase{
  public TestClassOne(String method){
    super(method);
}
public void testAddition(){
  assertEquals(4,2 + 2);
}
public void testSubtraction(){
  assertEquals(0,2 - 2);
}
}
```

默认的动作对这个类使用 Java 反射，将运行 testSubtraction()和 testAddition()。

现在假设有了第二个类 TestClassTwo。它使用了 brute-force 算法来寻找旅行销售商 Bob 的最短行程。关于旅行销售商人的算法的有趣事情是，当城市数目小的时候，它能工作正常，但是它是一个指数型的算法。比如，数百个城市的问题可能需要 20 000 年才能运行出结果。甚至 50 个城市都需要花上数小时的运行时间，因此，在默认情况下，可能不包括这些测试。

```
import junit.framework. * ;
public class TestClassTow extends TestCase{
  public TestClassTow(String method){
 super(method);
}
//This one takes a few hours...
public void testlongRunner(){
TSP tsp = new TSP();//Load with default cities
assertEquals(2300,tsp.shortestPath(50));//top 50
}
public void testShortTest(){
TSP tsp = new TSP();//Load with default cities
```

```
    assertEquals(140,tsp.shortestPath(5));//top 5
  }
  public void testAnotherShortTest(){
    TSP tsp = new TSP();//Load with default cities
    assertEquals(586,tsp.shortestPath(10));//top 10
  }
   public static Test suite(){
    TestSuite suite = new TestSuite();
     //only include short tests
   suite.addTest(
          new TestClassTow("testShortTest"));
   suite,addTest(
          new TestClassTow("testAnotherShortTest"));
    return suite;
    }
  }
```

要运行测试必须显示说明要运行它。没有这个特殊的机制，当调用 test suite 的时候，只有那些运行不花多少时间的测试才会被运行。

而且，此时看到了给构造函数的 String 参数是做什么用的了，它让 TestCase 返回了一个对命名测试方法的引用。这使用它来得到那两个耗时少的方法的引用，以把它们包含到 test suite 之中。

可能想要一个高一级别的测试来组合这两个测试类。

```
import joint.framework. * ;
public class TestClassComposite extends TestCase{
  public TestClassComposite(String method){
    super(method);
  }
  static public Test suite(){
      TestSuite suite = new TestSuite();
      //Grab everything:
      suite.addTestSuite(TestClassOne.class);
      //Use the suite method:
      suite.addTest(TestClassTow.suite());
      return suite;
  }
}
```

现在，如果运行 TestClassComposite，以下单个的测试方法都将被运行。

- 来自 TestClassOne 的 testAddition()
- 来自 TestClassOne 的 testSubtraction()
- 来自 TestClassTwo 的 testShortTest()
- 来自 TestClassTwo 的 testAnotherShortTest()

可以继续这种模式，另外一个类可能会包含 TestClassComposite，这将使得它包括上面所有

的测试方法，另外还会有它包含的其他测试的组合等。

1. Per-method 的 setUp 和 tearDown

每个测试的运行都应该是互相独立的；从而可以在任何时候，以任意的顺序运行每个单独的测试。

为了获得这样的好处，在每个测试开始之前，都需要重新设置某些测试环境，或者在测试完成之后，需要释放一些资源。JUnit 的 TestCase 基类提供以下两个方法供改写，分别用于环境的建立和清理。

```
protected void setUp();
protected void tearDown();
```

在以上例子中，在调用每个 test 方法之前，调用方法 setUp()；并且在每个测试方法完成之后，调用方法 tearDown()，如图 8.2 所示。

```
1.   oneTimeSetUp()                          Per-method
2.        setUp()                            setUp runs
3.     testMethod1()                         before each      One-time setUp
4.      tearDown()                           test method,     runs at start of
5.        setUp()                            and tearDown     suite,and tearDown
6.     testMethod2()                         runs after       runs at end of suite.
7.      tearDown()                           each method.
8. oneTimeTearDown()
```

图 8.2　代码的执行顺序

例如，假设对于每个测试，都需要某种数据库连接。这时，就不需要在每个测试方法中重复建立连接和释放连接，而只需在 setUp 和 tearDown 方法中分别建立和释放连接。

```
public class Test DB extends TestCase{
  private connection dbConn;
  protected void setUp(){
    dbConn = new Connection("oracle",1521,
                            "fred","foobar");
     dbConn.connect();
}
protected void tearDown(){
      dbConn.disconnect();
      dbConn = null;
 }
 public void testAccountAccess(){
   //Uses dbConn
  xxx xxx xxxxxx xxx xxxxxxxxx;
  xx xxx xxx xxxx x xx xxxx;
 }
 public void testEmployeeAccess(){
   //Uses ddbConn
   xxx xxx xxxxxx xxx xxxxxxxxx;
   xxxx x x xx xxx xx xxxx;
  }
}
```

在以上例子中，在调 testAccountAccess()之前，将会先调用 setUp()；然后在 testAccountAccess()完成之后，会接着调用 tearDown()。在第二个测试函数 testEmployeeAccess()中，也是按顺序先调用 setUp()，再调用该函数，最后调用 tearDown()。

2. Per-suite setUp 和 tearDown

一般而言，只需针对每个方法设置运行环境；但是在某些情况下，须为整个 test suite 设置一些环境，以及在 test-suite 中的所有方法都执行完成后做一些清理工作。要达到这种效果，需要 per-suite setUp 和 per-suite tearDown(就执行顺序而言，per-test 和 per-suite 之间的区别如图 8.2 所示)。

Per-suite 的 setUp 要复杂些，需要提供所需测试的一个 suite(无论通过什么样的方法)并且把它包装进一个 TestSetup 对象。使用前面的例子结果如下：

```
import junit.framework. * ;
import junit.extensions. * ;
public class TestClassTow extends TestCase{
  private static TSP tsp;
  public TestClassTow (String method){
   super(method);
}
//This one takes a few hours...
public void testLongRunner(){
 assertEquals(2300,tsp.shortestPath(50));
}
public void testShort Test(){
 assertEquals(140,tsp.shortestPath(5));
}
public void testAnotherShortTest(){
 assertEquals(586,tsp.shortestPath(10));
}
public static Test suite(){
 TestSuite suite = new TestSuite ();
 //only include short tests
 suite.addTest(new TestClassTow("testShortTest"));
 suite.addTest(new TestClassTow ("testAnotherShortTest"));
 TestSetup wrapper = new TestSetup(suite){
    protected void setUp(){
        OneTimeSetUp();
  }
  protected void tearDown(){
   oneTimeTearDown();
  }
};
return wrapper;
}
```

```
public static void onetimeSetUp(){
//one - time initialization code goes here...
tsp = new TSP();
tsp.loadCities("EasternSeaboard");
}
public static void oneTimeTearDown(){
//one - time cleanup code goes here...
  tsp.releaseCities();
}
}
```

注意,可以在同一个类中同时使用 per-suite 和 per-test 的 setUp()和 tearDown()。

8.3.5 自定义 JUnit 断言

通常而言,JUnit 所提供的标准断言对大多数测试已经足够了。然而,在某些环境下,譬如要处理一个特殊的数据类型,或者处理对多个测试都共享的一系列操作,如果有自定义的断言,将会更加方便。

在测试代码中,请不要复制和粘贴公有代码;测试代码的质量应该和产品代码一样,也就是说,在编写测试代码的时候,也应该维持好的编码原则,诸如 DRY 原则、正交性原则等。因此,需要把公共的测试代码抽取到方法中去,并且在测试用例中使用这些方法。

如果有需要在整个项目中共享的断言或者公共代码,需要考虑从 TestCase 继承一个类并且使用这个子类来进行所有的测试。例如,假使在测试一个经济方面的程序并且事实上所有的测试都使用了名为 Money 的数据类型,不直接从 TestCase 继承,相反创建了一个项目特有的基础测试类,如:

```
 import junit.framework.*;
 /**
   * project - wide base class for Testing
   */
public class ProjectTest extends TestCase{
   /**
     * Assert that the amount of money is an even
     * number of dollars (no cents)
     * @param message Text message to display if the
     *                              assertion fails
     * @param amount Money object to test
     *
     */
public void assertEvenDollars(String message,Money amount){
     assertEquals(message,
                    amount.asDouble() - (int)amount.asDouble(),
                    0.0,
                    0.001);
}
```

```
/ **
  * Assert that the amount of money is an even
  * number of dollars(no cents)
  * @param amount Money object to test
  *
  * /
public void assertEvenDollars (Money amount){
   assertEvenDollars("",amount);
}
}
```

在此，提供了两种形式的断言，一种接收一个 String 参数，另外一种则没有。注意，并没有复制和粘贴代码，而只是把第二个调用委托给了第一个。

现在，项目中的所有其他测试类将从这个基类继承下来而不是直接从 TestCase 进行继承：

```
public class TestSomething extends ProjectTest {
    …
```

事实上，开始新项目时总是从自己的自定义基类继承而不直接从 JUnit 的类继承，即便自己的基类在一开始没有添加任何额外的功能。这样做的好处是当需要添加一个所有测试类都需要的方法或者能力时，可以简单地编写自己的基类而不需要改动项目中的所有 test case。

8.3.6 JUnit 和异常

对于测试而言，下面两种异常是令人感兴趣的。

(1) 从测试代码抛出的可预测异常。

(2) 由于某个模块(或代码)发生严重错误而抛出的不可预测异常。

异常能够告诉人们什么东西出错了。有时，在一个测试中，需要被测试方法抛出一个异常，例如，有一个名为 sortMyList()的方法，如果传入参数是一个 null list，那么希望该方法抛出一个异常。在这种情况下，就需要显式的测试这一点。

```
Line 1   public void testForException(){
     - try{
     -     sortMyList(null);
     -     fail("Should have thrown an exception");
     5     } catch (RuntimeException e){
     -        assertTrue(true);
     -     }
     -   }
```

被测试的方法调用被第 3 行的 try/catch 块包含于内。预期中这个方法会抛出一个异常，如果异常如预期那样发生了，则代码将跳到第 6 行并且记录下断言以作统计目的使用。

现在可能要问为什么还要用 assertTrue 呢？它什么也不干，它不会失败，为什么还要把它放进来？任何对 assertTrue(true)的使用都应该被翻译为“预期控制流程会达到这个地方”，这对将来可能的误解来说会起到强有力的文档的作用。然而，不要忘记一个

assertTrue(true)没有被调用不会产生任何错误。

通常而言,对于方法中每个被期望的异常,都应该写一个专门的测试来确认该方法在应该抛出异常的时候确实会抛出异常。然而,这样虽然能够确认期望的异常,但是对于出乎意料的异常,应该怎么办呢?

虽然能够捕捉所有的异常并且调用 JUnit 的 fail(),但是最好让 JUnit 来做这件困难事。例如,假设正在读取一个包含测试数据的文件,不要自己去捕捉所有可能的 I/O 异常,而是简单地改变测试方法的声明让它能抛出可能的异常,如:

```
public void testData1()throws FileNotFoundException {
    FileInputStream in = new FileInputStream("data.txt");
    xxx xxx xxxxxx xxxxx xxxx;
}
```

实际上,JUnit 框架可以捕获任何异常,并且把它报告为一个错误,这些都不需要参与。更好的是,JUnit 不只是让一个断言失败,而是能够跟踪整个栈,并且报告 bug 的栈调用顺序,当需要查找一个失败测试的原因时,这将非常有用。

8.3.7 关于命名的更多说明

通常而言,都希望所有的测试在任何时候都能够顺利通过。但假设之前想到了一些测试,并且编写了这些测试,现在正在编写能够通过测试的实现代码。那么这些还不具备实现代码的新测试未能通过,又该怎么办?

虽然可以继续编写这些测试,但现在却不能让测试框架运行这些测试。幸运的是大部分测试框架使用了命名习惯来自动发现测试。比如,当用 Java 使用 JUnit 时,以 test 开头的方法(比如 testMyThing)将作为测试来运行,所有需要做的事情就是把方法命名为别的,然后要来运行它时再改回来。如果把进行中的测试命名为 pendingTestMyThing,那么不仅测试框架现在会忽略它,而且还能通过在所有代码中搜索字符串 pendingTest 来轻易寻找到漏掉的所有测试。当然,代码必须能编译通过,如果不能,那么应当注释掉那些无法编译的部分。要避免养成忽略"失败的测试结果"的习惯。

8.3.8 JUnit 测试骨架

用 JUnit 写测试真正所需要的就 3 件事:

(1) 一个 import 语句引入所有 junit.framework.* 下的类。

(2) 一个 extends 语句让类从 TestCase 继承。

(3) 一个调用 super(string)的构造函数。

许多 IDE 会提供这些,这样写出来的类能够使用 JUnit 的 test runner 运行,并且自动执行类中所有 test 方法。

但是有时不从 JUnit 的 runner 运行,而是直接运行一个测试类,这样会更方便一些。每个测试运行前和后的方法名又是什么?

可以制作一个骨架来提供所有这些特性并且做得相当简单。

现在,已经知道了如何编写测试,接下来是进一步介绍找出哪些需要测试的内容。

8.4　测试的内容

1. 结果是否正确

对于测试而言,首要的也是最明显的任务就是查看所期望的结果是否正确——验证结果。

通常一些简单的测试,甚至这类测试的一部分在需求说明中都已经指定了。如果文档中没有的话,那么就需要问其他人员。总之,必须能够最终回答这个关键问题。

如果代码能够运行正确,要怎么才知道它是正确的呢? 如果不能很好的回答这个问题,那么编写代码或者测试完全就是在浪费时间。试想,如果文档比较晦涩或者不完整的话,该怎么办呢? 这是否意味着不能编写代码,而必须等到文档都已经齐备且清楚时才能继续编写代码呢? 完全不是。如果文档还不明了,或者不完整的话,至少总是可以自己发明出一些需求来。虽然从用户的角度来看,这些功能或许是不准确的,但是现在就可以知道编写的代码要做什么,从而能够回答上面的问题。

当然,必须安排一些来自用户的反馈以调整自己的假设。在代码的整个生命期中,“正确”的定义可能会不断在变;但是无论如何,至少需要确认代码所做的和自己的期望是一致的。

对于许多有大量测试数据的测试,可能会考虑用一个独立的数据文件来存储这些测试数据,然后让单元测试读取该文件。这并不困难,甚至并不需要使用 XML 文件。以下代码是 TestLargest 的一个版本,它从一个测试文件中读取所有的测试数据。

```
import  junit.framework.*;
import  java.io.*;
import  java.util.ArrayList;
import  java.util.StringTokenizer;
public class TestLargestDataFile extends TestCase {
  public TestLargestDataFile(String name){
    super(name);
    }
    /* Run all the tests in testdata.txt(does not test
     * exception case). We'll get an error if any of the
     * file I/O goes wrong.
     */
    public void testFromFile()throws Exception {
      String line;
      BufferedReader rdr = new BufferedReader(
                                  new FileReader(
                                  "testdata.txt"));
    while((line = rdr.readLine())!= null){
     if(line.startsWith("#")){// Ignore comments continue;
    }
    StringTokenizer st = new StringTokenizer(line);
    If (!st.hasMoreTokens()){
```

```
      Continue;// Blank line
    }
    //Get the expected value
    String val = st.nextToken();
    int expected = Integer.valueof(val).intValue();

    //And the arguments to Largest
    ArrayList argument_list = new ArrayList();
    while (st.hasMoreTokens()){
     argument_list.add(Integer.valueof(st.nextToken()));
    }
    //Transfer object list into native array
    int[] arguments = new int [argument_list.size()];
    for (int i = 0;i<argument_list.size();i++){
      arguments[i] = ((Integer)argument_list.
                           get(i)).intValue();
    }
    //And run the assert
      assertEquals(expected,
                       Largest.largest(arguments));
    }
  }
}
```

数据文件的格式很简单,每行一些数字,其中第一个数字是期望的答案,剩余的数字就是要用来测试的参数。另外,使用井号(#)来表示所在行是注释,因此可以在测试文件中添加一些有意义的注释或者描述。

测试文件的具体形式如下:

```
#
#Simple tests;
#
9 7 8 9
9 9 8 7
9 9 8 9
#
# Negative number tests;
#
-7 -7 -8 -9
-7 -8 -7 -8
-7 -9 -7 -8
#
#Mixture;
#
7 -9 -7 -8 7 6 4
9 -1 0 9 -7 4
```

```
#
# Boundary conditions;
#
1 1
0 0
2147483647 2147483647
-2147483648 -2147483648
```

如上面的例子一样只有很少的东西要测试,就不值得费这么大劲了。但是假如面对的是一个很复杂的应用程序,而表格中有几百个甚至几千个测试数据,那么测试文件就是一个很有吸引力的选择。

多注意一下测试数据,不管文件中的还是代码中的测试数据,都有可能是不正确的。实际经验告诉我们,测试数据比代码更有可能是错的,特别是人工计算的,或者来自原由系统计算结果的测试数据(系统添加的新特性,可能故意导致了不同结果)。因此,当测试数据显示有错误发生的时候,应该在怀疑代码前先对测试数据检查两三遍。

另外,还有一些值得考虑的,代码本身是否并没有测试任何异常的情况,要实现这个功能,需要怎么来做?

一个原则是对于验证被测方法是正确的这件事情,如果某些做法能够使它变得更加容易,那么就采纳它。

2. 边界条件

在前面"求最大值"的例子中,发现了几个边界条件,最大值位于数组末尾,数组包含负数,或者数组为空等。

找边界条件是做单元测试中最有价值的工作之一,因为 bug 一般就出现在边界上。下面是一些需要考虑的条件。

- 完全伪造或者不一致的输入数据,例如一个名为"!* W:X\&Gi/W~>g/h# WQ@"的文件。
- 格式错误的数据,例如没有顶层域名的电子邮件地址,就像 fred@foobar 这样的。
- 空值或者不完整的值(如 0,0.0,""和 null)。
- 一些与意料中的合理值相去甚远的数值。例如一个人的岁数为 10 000 岁。
- 如果要求的是一个不允许出现的重复数值的 list,但是传入的是一个存在重复的数值的 list。
- 如果要求的是一个有序的 list,但传入的是一个无序的 list,或者反之。例如,给一个要求排好序的算法传入一个未排序的 list,甚至一个反序的 list。
- 事情到达的次序是错误的,或者碰巧和期望的次序不一致。例如,在未登录系统之前,就尝试打印文档。

一个想到可能的边界条件的简单办法就是记住助记短语 CORRECT。对于其中的每一条,都应该想想它是否与存在于被测方法中的某个条件非常类似,而当这些条件被违反时,出现的又是什么情形。

- Conformance(一致性)——值是否和预期的一致。
- Ordering(顺序性)——值是否如应该的那样,是有序或者无序的。

- Range(区间性)——值是否位于合理的最小值和最大值之内。
- Reference(依赖性)——代码是否引用了一些不在代码范围之内的外部资源。
- Existence(存在性)——值是否存在(例如,是否非 null,非 0,在一个集合中等)。
- Cardinatity(基数性)——是否恰好有足够的值。
- Time(相对或者绝对的时间性)——所有事情的发生是否是有序的?是否是在正确的时刻?是否恰好及时?

3. 检查反向关联

对于一些方法,可以使用反向的逻辑关系来验证它们。例如,可以用对结果进行平方的方式来检查一个计算平方根的函数,然后测试结果是否和原数据很接近;

```
public void testSquareRootUsingInverse()
{
  double x = mySquareRoot(4.0);
  assertEquals(4.0,x * x,0.0001);
}
```

类似地,为了检查某条记录是否成功地插入了数据库,也可以通过查询这条记录来验证等。

要注意的是:当同时编写了原方法和它的反向测试时,一些 bug 可能会被两个函数中都出现的错误所掩盖。在可能的情况下,应该使用不同的原理来编写反向测试。在上面平方根的例子中,用的只是普通的乘法来验证。而在数据库查找的例子中,大概可以使用厂商提供的查找方法来测试自己的插入。

4. 使用其他手段来实现交叉检查

同样可以使用其他手段来交叉检查函数的结果。

通常而言,计算一个量会有一种以上的算法,可能会基于运行效率或者其他的特性来选择算法,那是要在产品中使用的。但是在测试用的系统中,可以使用剩下算法中的一个来交叉测试结果。当确实存在一种经过验证并能完成任务的算法,只是由于速度太慢或者太不灵活而没有在产品代码中使用时,这种交叉检查的技术非常有效。

可充分利用一些比较弱的版本来检查新写的较好的版本,看它们是否产生了相同的结果,如:

```
public void testSquareRootUsingStd(){
  double number = 3880900.0;
  double root1 = mySquareRoot (number);
  double root2 = Math.sqrt(number);
  assertEquals(root2,root1,0.0001);
}
```

另外一种办法是,使用类本身不同组成部分的数据,并且确信它们能"合起来"。例如,假设正在做一个图书馆的数据系统。在这个系统中,对于每一本具体的书,它的数量永远是平衡的。也就是说,借出的数加上躺在架子上的库存数应当永远等于总共所藏的书籍数量,这些就是数据的两个分开的不同组成部分(借出数和库存数)。它们甚至可以由不同类的对象来汇报它们,但是它们仍然必须遵循上面的约束(即平衡,总数恒定)。因而可以在它们之

间进行交叉检查，即用一种数量检查另一种数量。

5. 强制产生错误条件

在真实世界中，错误总是会发生的：磁盘满、网络连线断开、电子邮件过多、程序崩溃。应当能够通过强制引发错误，来测试自己的代码是如何处理所有这些真实世界中的问题。

下面是一些所能想到的可以用来测试函数的环境方面的因素：内存耗光、磁盘用满、时钟出问题、网络不可用或者有问题、系统过载、调色板颜色数目有限和显示分辨率过高或者过低。

6. 性能特性

一个检查起来会很有益处的部分是性能特性，而不是性能本身。然而性能特性却有一种类似于“随着输入尺寸慢慢变大，问题慢慢变复杂”的趋势。

我们想要的是一个性能特性的快速回归测试。很多时候，也许发布的第一个版本工作正常，但是第二个版本不知道为何变得很慢。

为了避免这种尴尬的场景发生，可以考虑实现一些粗糙测试来确保性能曲线能够保持稳定。例如，假设已经编写了一个过滤器，它能够鉴别希望阻止的 Web 站点。

那段代码在几十个样板站点上都工作正常，但要是 10 000 个呢？100 000 个呢？先写点单元测试来查看，代码如下。

```
public void testURLFilter(){
  Timer timer = new Timer();
  String naughty_url = "http://www.xxxxxxxx..com;"
//First,check a bad URL against a small list
URLFilter filter = new URLFilter(small_list);
timer.start();
filter.check(naughty_url);
timer.end();
assertTrue(timer.elapsedTime()<1.0);

//Next,check a bad URL against a big list
URLFilter f = new URLFilter(big_list);

time.start();
fliter.check(naughty_url)
timer.end();

assertTrue(timer.elapsedTime()<2.0);

//Finally,check a bad URL against a huge list
URLFilter f = new URLFilter(huge_list);

timer.start();
filter.check(naughty_url);
timer.end();
```

```
assertTure(timer.elapsedTime()<3.0);
}
```

这保证了性能方面的要求，但是运行这一个测试就花去了 6～7 秒钟，所以可能不想每次都运行它。因此，只要每晚或者每隔几天运行它一次，就能快速地定位到可能引入的任何问题，而此时仍然有时间来修正它们。

可能需要一些测试辅助工具，它们能够提供对单个测试进行计时，模拟高负载情况之类的功能，比如免费的 JUnitPerf。

8.5 JUnit 测试实例

通过前面的学习，已经掌握了 JUnit 的基本使用方法，下面利用它对一个具体的实例进行测试。

本例使用 Eclipse 中的 JUnit 工具建立测试。打开 Eclipse，建立一个新的工程的工作空间，输入工程名称，比如 ProjectWithJUnit，单击完成。这样就建立了一个新工程，配置一下 Eclipse，把 JUnit library 添加到 build path，执行【Project】→【Properties】命令，选择【Java Build Path Libraries】，再单击【Add Exteranal JARs】选中【JUnit.jar】，可以看到 JUnit 将会出现在屏幕上【libraries】列表中，单击【OK】按钮，Eclipse 将强制 rebuild 所有的 buildpaths。

为了方便，假定将要写的类名是 HelloWorld，有一个返回字符串的方法 say()。建立这样一个 test，在 ProjectWithJUnit 标题上右击，选择【New】→【Other】命令，展开【Java】，选择 JUnit 里面的【JUnit Test Case】选项，接着单击【Next】按钮，如图 8.3 所示。

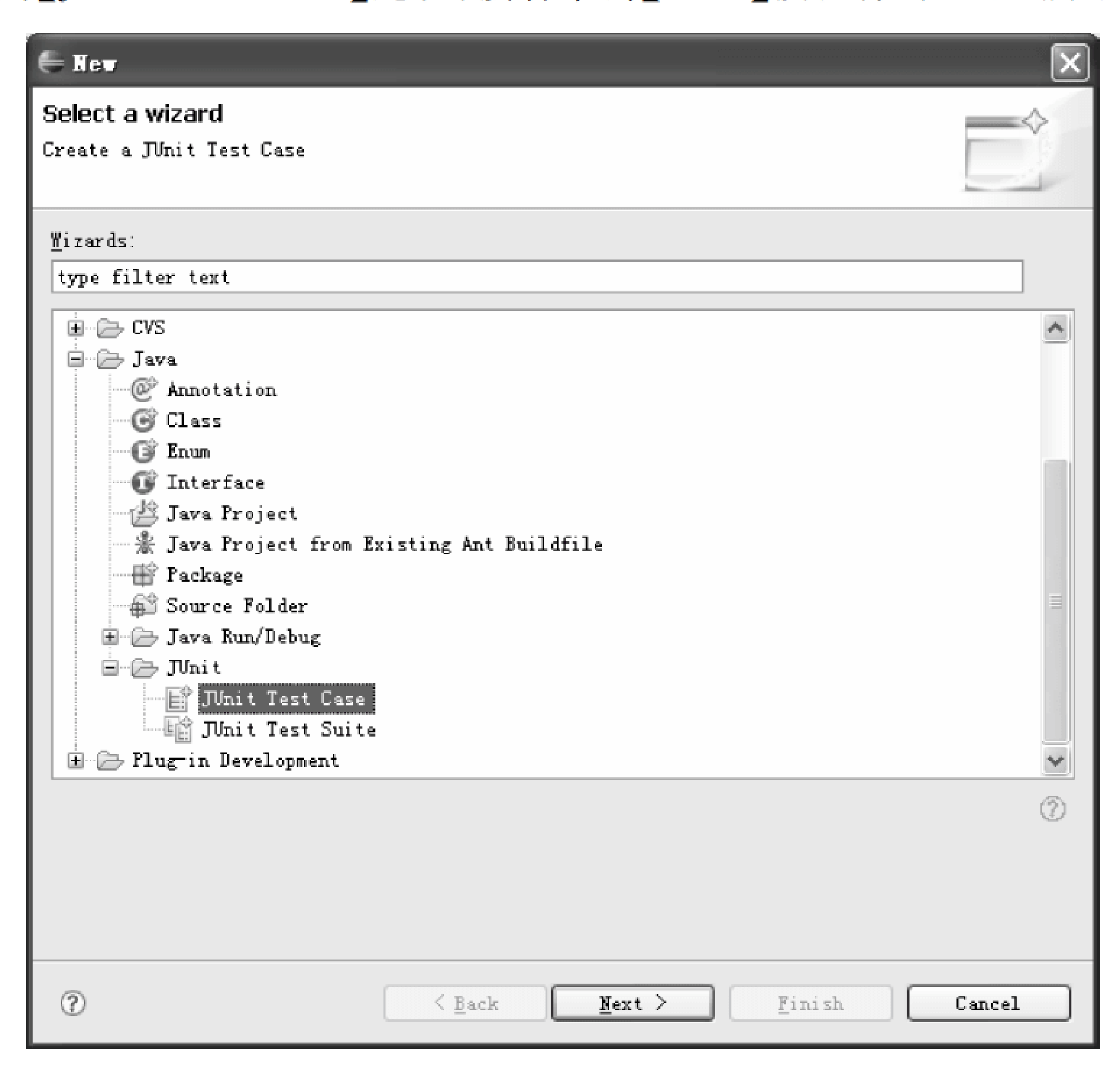


图 8.3　在 Eclipse 中建立 JUnit test

在 Class under test 一栏里输入需要测试的类名 HelloWorld。接下来，在工程 ProjectWithJUnit 中新建一个名为 TestThatWeGetHelloWorldPrompt 的类，用来测试类 HelloWorld，单击【Finish】按钮完成。

下面是 TestThatWeGetHelloWorldPrompt.java 的代码。

```
public class TestThatWeGetHelloWorldPrompt extends TestCase
{
    public TestThatWeGetHelloWorldPrompt(String name)
    {
        super(name);
    }
    public void testSay()
    {
        HelloWorld hi = new HelloWorld();
        assertEquals("Hello World!",hi.say());
    }
    public static void main(String[] args)
    {
        junit.textui.TestRunner.run(TestThatWeGetHelloWorldPrompt.class);
    }
}
```

这个代码继承了 JUnit 的 TestCase，TestCase 在 JUnit 的 javadoc 里的定义是用来运行多个 Test 的固定装置。JUnit 也定义了 TestSuite 由一组关联的 TestCase 组成。

通过以下两步来建立简单的 Test Case。

(1) 建立 JUnit.framework.TestCase 的实例。

(2) 定义一些以 test 开头的测试函数，并且返回一空值。

TestThatWeGetHelloWorldPrompt.java 同时遵循这些标准。这些 TestCase 的子类含有一个 testSay()的方法。这个方法由 assertEquals()方法调用，用于检验 say()的返回值。

主函数 main()用来运行 test 并且显示输出的结果。JUnit 的 TestRunnery 以图形(swing.ui)和本文(text.ui)的方式来执行 test 并反馈信息。使用文本(text.ui)方式 Eclipse 肯定支持。所谓文本和图形，是指建立 TestCase 的时候，有一个选项【Which method stubs would you like to create】，选择【text.ui ‖ swing.ui ‖ awt.ui】，一般是选择 text.ui。依照这些文本的信息，Eclipse 同时会生成图形显示。

一旦运行了 test，应该看到返回一些错误的信息。执行【Run】→【Run as】→【JUnit Test】命令，可以看到 JUnit 窗口会显示出一个红色条，表示是一个失败的 test，如图 8.4 所示。

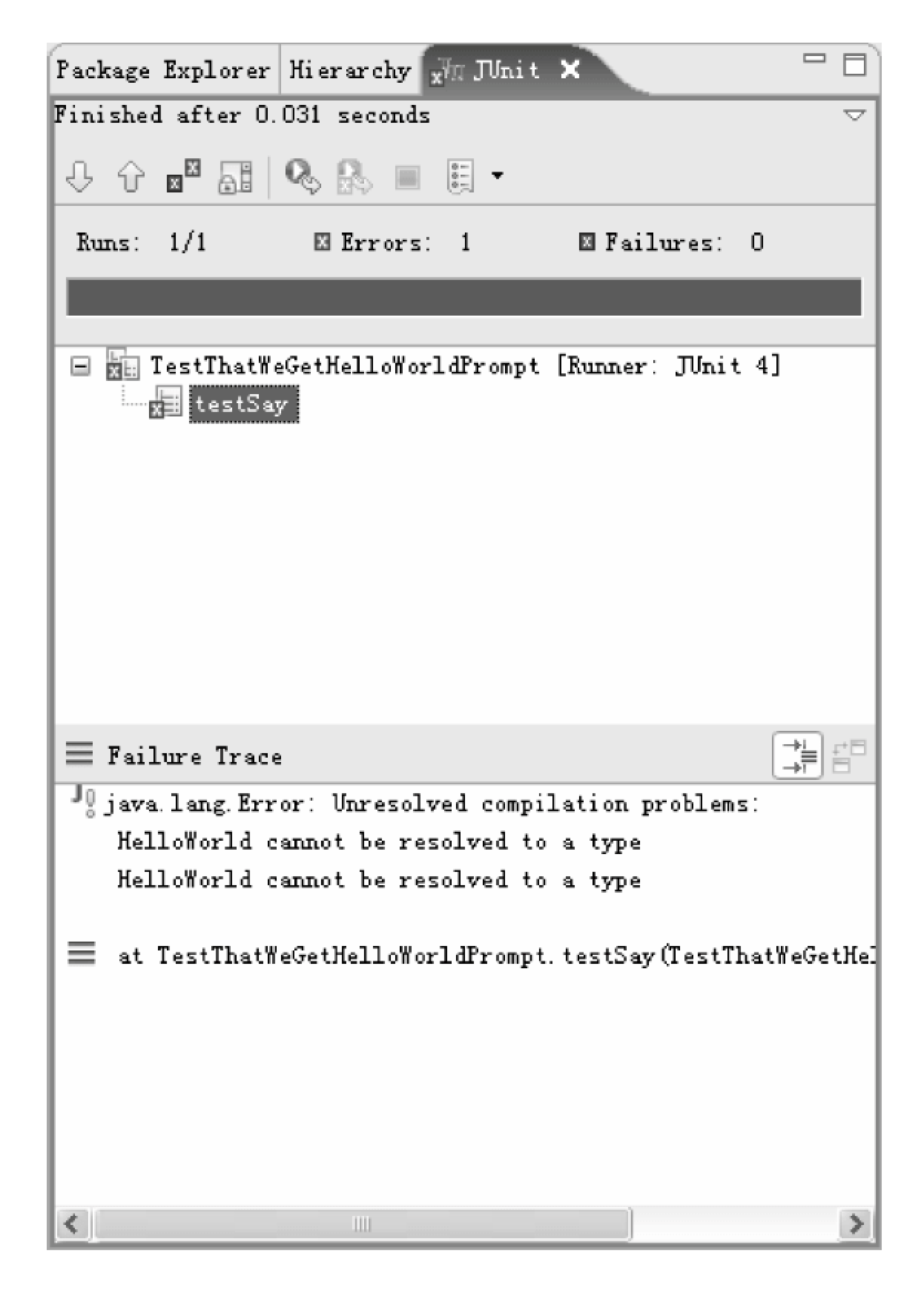


图 8.4　JUnit 窗口显示第一次测试失败

现在正式开始建立用于工作的 HelloWorld 代码，执行【New】→【Class】命令，代码如下：

```
public class HelloWorld
{
    public String say()
    {
        return("Hello World!");
    }
}
```

再来测试一下看看结果，执行【Run】→【Run As JUnit】命令，在 JUnit 窗口中出现了一个绿条，表示测试通过，如图 8.5 所示。

现在，再改变一个条件让测试不通过。这将帮助我们理解 JUnit test 是怎样覆盖并且报出不同错误的。编辑 assertEquals()方法，把它的返回值从"Hello World!"变成另外一个值，比如"Hello ME!"。这样，再运行这个 JUnit test，显示条又变成红色，并且可以看到是什么原因导致的错误，如图 8.6 所示。

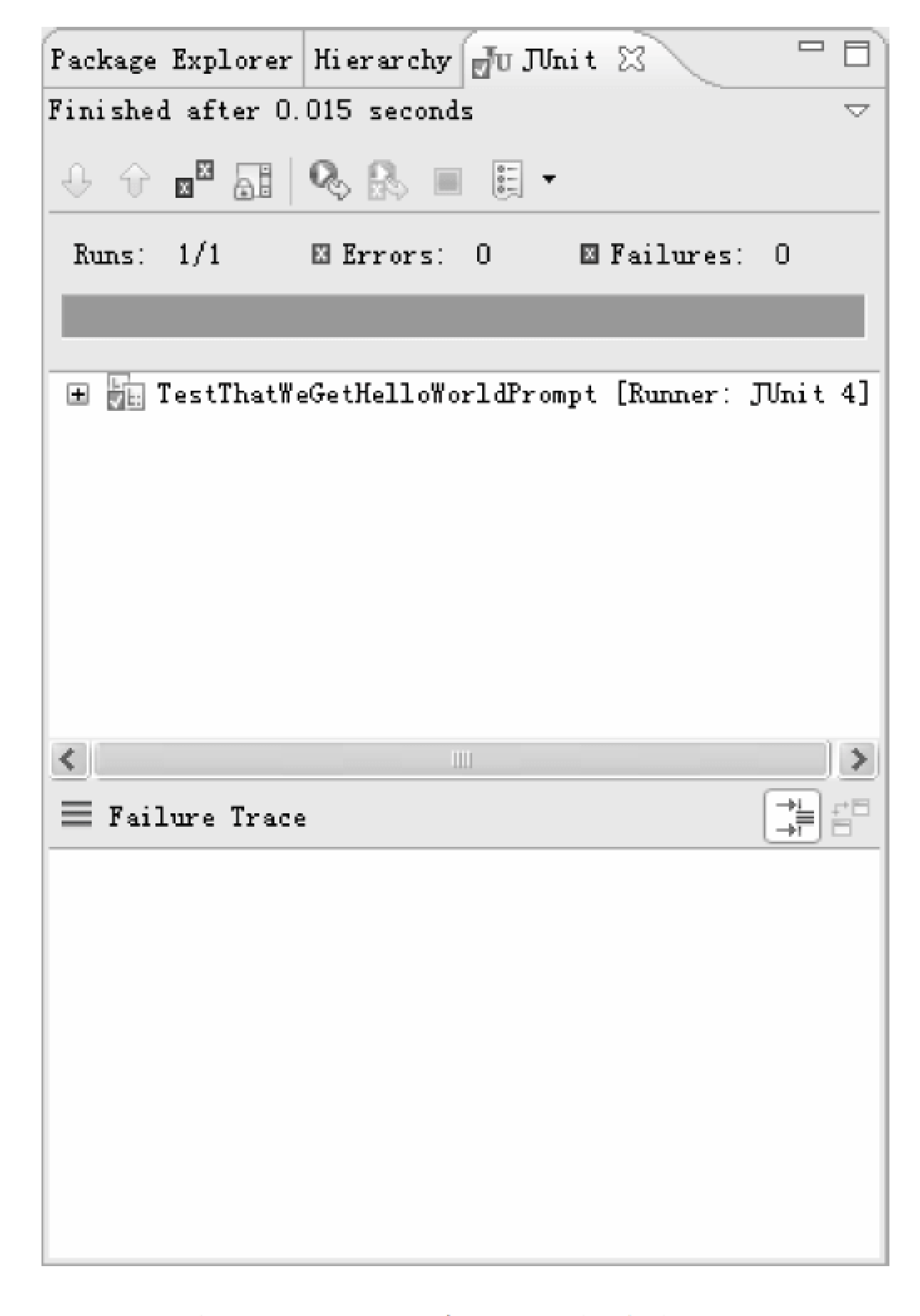



图 8.5　JUnit 窗口显示测试通过

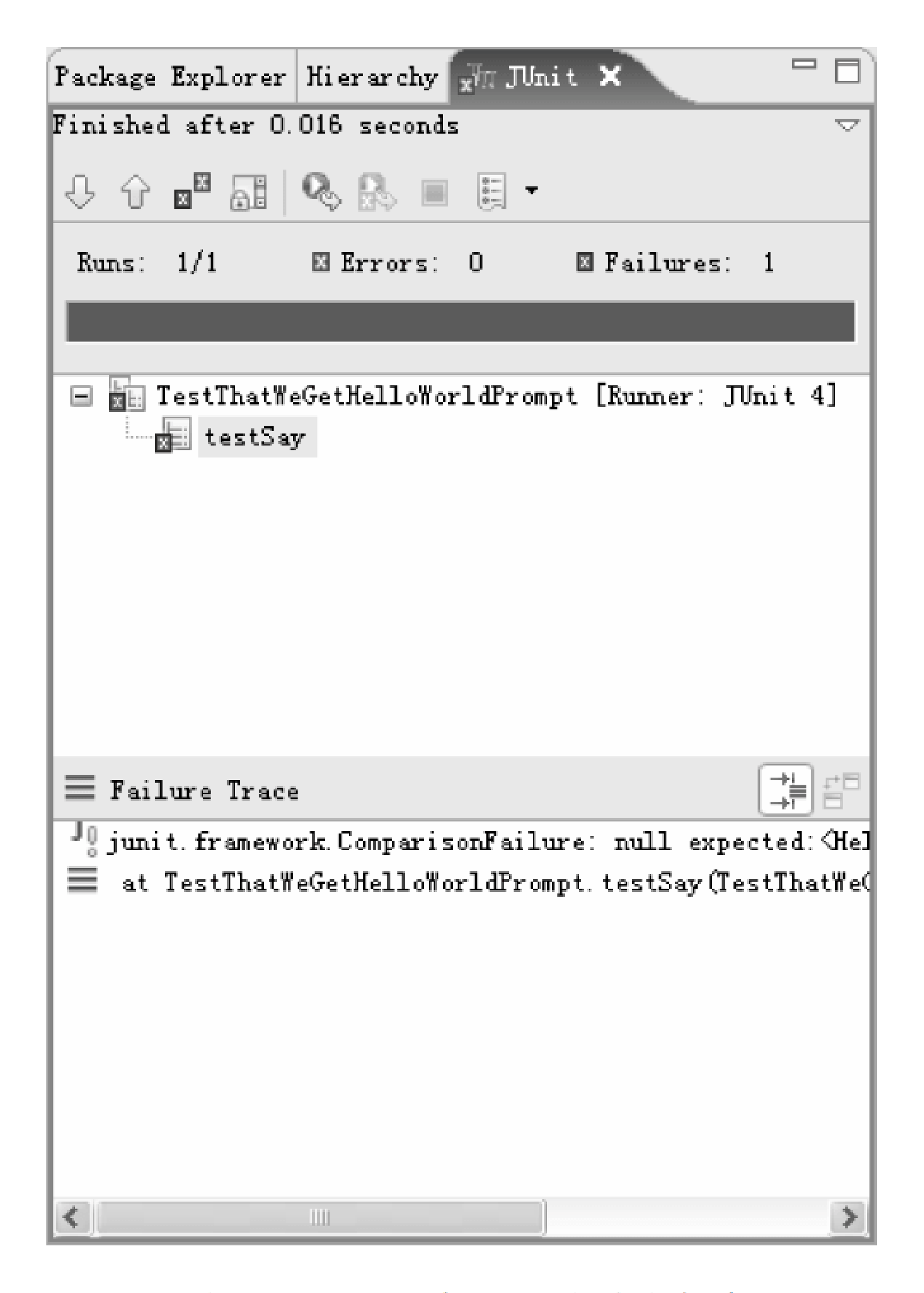


图 8.6　JUnit 窗口显示测试失败

练　习　题

1. 简述 JUnit 单元测试步骤。
2. 对下列代码进行单元测试。

```
Triangle.java
public class Triangle
{
  //定义三角形的三边
  protected long lborderA = 0;
  protected long lborderB = 0;
  protected long lborderC = 0;
  //构造函数
  public Triangle(long lborderA,long lborderB,long lborderC)
  {
    this.lborderA = lborderA;
    this.lborderB = lborderB;
    this.lborderC = lborderC;
  }
  /*
  * 判断是否是三角形
```

```
 */
public boolean isTriangle(Triangle triangle)
{
  boolean isTrue = false;
  //判断边界是否大于 0 小于 200,出界返回 false
  if((triangle.lborderA>0&&triangle.lborderA<200)
    &&(triangle.lborderB>0&&triangle.lborderB<200)
    &&(triangle.lborderC>0&&triangle.lborderC<200))
    {
      //判断两边之和是否大于第三边
      if((triangle.lborderA<(triangle.lborderB + triangle.lborderC))
        &&(triangle.lborderB<(triangle.lborderA + triangle.lborderC))
        &&(triangle.lborderC<(triangle.lborderA + triangle.lborderB)))
      isTrue = true;
     }
    return isTrue;
  }
/*
 * 判断三角形类型
 */
public String isType(Triangle triangle)
{
  String strType = "";
  //判断是否是三角形
  if(this.isTriangle(triangle))
  {
    //判断是否是等边三角形
    if(triangle.lborderA == triangle.lborderB
      &&triangle.lborderB == triangle.lborderC)
      strType = "等边三角形";
    //判断是否是不等边三角形
    else if((triangle.lborderA! = triangle.lborderB)&&
            (triangle.lborderB! = triangle.lborderC)&&
            (triangle.lborderA! = triangle.lborderC))
             strType = "不等边三角形";
        else
            strType = "等腰三角形";
    }
   return strType;
  }
}
```

参 考 文 献

[1] (美)乔根森(Jorgensen,P. C.). 软件测试(第 2 版). 韩柯译. 北京:机械工业出版社,2003

[2] 贺平. 软件测试教程. 北京:电子工业出版社,2005

[3] 朱少民. 软件测试方法和技术. 北京:清华大学出版社,2005

[4] (美)托马斯(Thomas,D.). 单元测试之道 Java 版:使用 JUnit. 陈伟柱译. 北京:电子工业出版社,2005

[5] (美)佩里(Perry,W. E.). 软件测试的有效方法(第 2 版). 兰雨晴译. 北京:机械工业出版社,2004

[6] (美)凯纳(Kaner,C.). 软件测试经验与教训. 韩柯译. 北京:机械工业出版社,2004

[7] (美)维他科(Whittaker,J. A.). 实用软件测试指南. 马良荔译. 北京:电子工业出版社,2003

[8] (美)帕顿(Patton,R.). 软件测试. 周予滨译. 北京:机械工业出版社,2002

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602 室　计算机与信息分社营销室　收

邮编：100084　　　　电子邮件：jsjjc@tup.tsinghua.edu.cn

电话：010-62770175-4608/4409　　　　邮购电话：010-62786544

教材名称：软件测试技术基础

ISBN：978-7-302-17493-6

个人资料

姓名：__________　年龄：______所在院校/专业：__________

文化程度：________　通信地址：________________

联系电话：________　电子信箱：________________

您使用本书是作为：□指定教材 □选用教材 □辅导教材 □自学教材

您对本书封面设计的满意度：

□很满意 □满意 □一般 □不满意　改进建议____________

您对本书印刷质量的满意度：

□很满意 □满意 □一般 □不满意　改进建议____________

您对本书的总体满意度：

从语言质量角度看　□很满意 □满意 □一般 □不满意

从科技含量角度看　□很满意 □满意 □一般 □不满意

本书最令您满意的是：

□指导明确 □内容充实 □讲解详尽 □实例丰富

您认为本书在哪些地方应进行修改？（可附页）

____

____

您希望本书在哪些方面进行改进？（可附页）

____

____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。

"21世纪高等学校计算机教育实用规划教材"
系列书目

书　名	作　者	ISBN 号
32 位微型计算机原理·接口技术及其应用(第 2 版)	史新福等	9787302134039
AutoCAD 实用教程(配光盘)	张强华等	9787302127260
Internet 实用教程——技术基础及实践	田力	9787302110668
Java 程序设计实践教程	张思民	9787302132585
Java 程序设计实用教程	胡伏湘等	9787302109600
Java 语言程序设计	张思民	9787302144113
Visual Basic 程序设计基础	李书琴等	9787302132684
Visual C++ 程序设计与应用教程	马石安等	9787302155027
XML 实用技术教程	顾兵	9787302142867
大学计算机公共基础	阮文江	9787302143307
大学计算机网络公共基础教程	徐祥征等	9787302130161
大学计算机基础	刘腾红	9787302155812
大学计算机基础实验指导	刘腾红	9787302155522
大学计算机基础应用教程	黄强	9787302152163
多媒体技术教程——案例　训练与课程设计	胡伏湘等	9787302126201
多媒体课件制作——Authorware 实例教程	唐前军等	9787302156000
多媒体技术与应用	李飞等	9787302161653
汇编语言程序设计教程与实验	徐爱芸	9787302143413
计算机操作系统	颜彬等	9787302141471
计算机网络实用教程——技术基础与实践	刘四清等	9787302104513
计算机网络应用技术教程	孙践知	9787302118893
计算机网络与 Internet 实用教程——技术基础与实践	徐祥征等	9787302106593
计算机网络应用与实验教程	徐小明等	9787302158813
计算机硬件技术基础	张钧良	9787302160564
实用软件工程	陆惠恩	9787302125594
软件测试技术基础	陈汶滨等	9787302174936
软件工程技术与应用	顾春华等	9787302161318
软件开发技术与应用	李昌武等	9787302161257
数据库及其应用系统开发(Access 2003)	张迎新	9787302128281
数据库技术与应用——SQL Server	刘卫国等	9787302143673
数据库技术与应用实践教程——SQL Server	严晖等	9787302142317
数据库应用案例教程(Access)	周安宁等	9787302146056
数据库技术与应用	史令等	9787302161608

数据库原理及开发应用	周屹	9787302156802
数据库原理与 DB2 应用教程	杨鑫华等	9787302155546
网络技术应用教程	梁维娜等	9787302134848
网页制作教程	夏宏等	9787302105916
微型计算机原理及应用导教·导学·导考(第 2 版)	史新福等	9787302133995
程序设计语言——C	王珊珊等	9787302158035
PHP Web 程序设计教程与实验	徐辉等	9787302155508
面向对象程序设计教程(C++ 语言描述)	马石安等	9787302150534